



NAVAL POSTGRADUATE SCHOOL

MONTEREY CALIFORNIA

THESIS

**UTILIZATION OF WEB SERVICES TO IMPROVE
COMMUNICATION OF OPERATIONAL INFORMATION**

by

David Lowery

September 2004

Thesis Advisor:
Co-Advisor:

Don Brutzman
Curtis Blais

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2004	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Utilization of Web Services to Improve Communication of Operational Information			5. FUNDING NUMBERS	
6. AUTHOR David S. Lowery				
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS NSA, Ft. Meade, Maryland			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT <p>Currently under development, the Global Information Grid (GIG) Enterprise Services (ES) is a suite of capabilities intended to provide improved user access to mission-critical data via Web-based and network technologies. Some of the problems of implementing such capabilities include non-uniform data formats, incompatible run-time environments and nonstandard proprietary applications, all of which block operational interoperability.</p> <p>Web services are specifically designed to address the interoperability challenges of a service-oriented architecture (SOA) such as the GIG. SOAs are networked infrastructures that are designed to facilitate the interoperability of collections of services without requiring service context awareness. Standards-based Web services provide the necessary flexibility and extensibility to ensure information flow is platform, run-time and software independent.</p> <p>The proof of concept (POC) software example developed for this research demonstrates the flexibility and extensibility of standards-based, operating-system-independent Web services. The result is an experimental endeavor to provide a mock operation command center information portal, which provides a notional summary personnel status report to the commander in real-time from a Web service that was originally generated by a stand-alone client/server system. The POC is developed with great attention to open-source technologies and open-standards compliance. The key technologies involved are Extensible Markup Language (XML), the Java programming language, PHP: Hypertext Preprocessor (PHP) scripting language and Simple Object Access Protocol (SOAP).</p> <p>This work demonstrates the benefits of leveraging Web services to unlock legacy specialized applications to enhance the Warfighter's battlespace awareness by improving information flow via a Web based information portal.</p>				
14. SUBJECT TERMS Extensible Markup Language, XML, Web Services, Java Web Services, PHP, Extensible Stylesheet Language Transformation, XSLT, Web Portal, Common Operating Picture, COP, Global Information Grid, Enterprise Services, GIG, ES, Network Centric Warfare, W3C, World Wide Web Consortium			15. NUMBER OF PAGES 133	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

**UTILIZATION OF WEB SERVICES TO IMPROVE
COMMUNICATION OF OPERATIONAL INFORMATION**

David S. Lowery
Captain, United States Marine Corps
B.S., Valdosta State University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2004**

Author: David S. Lowery

Approved by: Don Brutzman
Thesis Advisor

Research Associate Curtis Blais
Thesis Co-Advisor

Dr. Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Currently under development, the Global Information Grid (GIG) Enterprise Services (ES) is a suite of capabilities intended to provide improved user access to mission-critical data via Web-based and network technologies. Some of the problems of implementing such capabilities include non-uniform data formats, incompatible run-time environments and nonstandard proprietary applications, all of which block operational interoperability.

Web services are specifically designed to address the interoperability challenges of a service-oriented architecture (SOA) such as the GIG. SOAs are networked infrastructures that are designed to facilitate the interoperability of collections of services without requiring service context awareness. Standards-based Web services provide the necessary flexibility and extensibility to ensure information flow is platform, run-time and software independent.

The proof of concept (POC) software example developed for this research demonstrates the flexibility and extensibility of standards-based, operating-system-independent Web services. The result is an experimental endeavor to provide a mock operation command center information portal, which provides a notional summary personnel status report to the commander in real-time from a Web service that was originally generated by a stand-alone client/server system. The POC is developed with great attention to open-source technologies and open-standards compliance. The key technologies involved are Extensible Markup Language (XML), the Java programming language, PHP: Hypertext Preprocessor (PHP) scripting language and Simple Object Access Protocol (SOAP).

This work demonstrates the benefits of leveraging Web services to unlock legacy specialized applications to enhance the Warfighter's battlespace awareness by improving information flow via a Web based information portal.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	OVERVIEW.....	1
B.	PROBLEM SPACE.....	2
C.	MOTIVATION.....	3
1.	Personal Background.....	3
2.	The Need for Improved Integration and Extensibility.....	4
3.	The Need for Web-based Interoperable Solutions.....	5
D.	THESIS ORGANIZATION.....	5
II.	RELATED WORK.....	7
A.	INTRODUCTION.....	7
B.	CURRENT OPEN SOURCE WEB SERVICE STANDARDS ORGANIZATIONS.....	7
1.	World Wide Web Consortium (W3C).....	7
2.	Web Services Interoperability (WS-I) Organization.....	9
3.	Organization for the Advancement of Structured Information Standards (OASIS).....	9
C.	JAVA-BASED WEB SERVICES.....	10
D.	BRIEF TECHNICAL ASSESSMENT OF WEB SERVICES.....	11
1.	Problem Resolution Through Maturing Standards and Technology.....	11
2.	Defining Web Services.....	12
3.	Best Practices.....	12
4.	Extensibility is Crucial.....	13
5.	Web Service Usage.....	13
E.	SUMMARY.....	14
III.	THE GLOBAL INFORMATION GRID (GIG).....	15
A.	INTRODUCTION.....	15
B.	THE CONCEPT OF GRID COMPUTING.....	15
C.	A JOINT WARFIGHTING PERSPECTIVE ON THE CONCEPT OF NETWORK-CENTRIC WARFARE.....	16
D.	GIG COMPOSITION.....	18
1.	Global Information Grid Enterprise Services (GIG ES).....	20
2.	Multiple Services Provided by a Single System.....	21
E.	THE WARFIGHTER'S PERSPECTIVE.....	23
F.	SUMMARY.....	26
IV.	DEVELOPMENT OF THE PROOF OF CONCEPT (POC) EXEMPLAR.....	27
A.	INTRODUCTION.....	27
B.	RESEARCH AND DEVELOPMENT METHODOLOGY.....	27
C.	DEVELOPMENT AND TESTING PLATFORMS.....	28
D.	DATA REQUIREMENTS.....	30

1.	Personnel Reporting (G1).....	30
E.	WEB SERVICE IMPLEMENTATION	30
1.	Service Providers	30
a.	Client Application	31
b.	Server Application.....	35
c.	Server Web Service.....	39
d.	Parser Web Service	42
e.	Transformation Web Service	43
2.	Service Requesters	43
a.	Basic Client	44
b.	Parser Client.....	45
c.	Transformation Client	46
F.	WEB PORTAL/REPORT IMPLEMENTATION.....	47
G.	SUMMARY	48
V.	PROOF OF CONCEPT (POC) EXEMPLAR RESULTS	49
A.	INTRODUCTION.....	49
B.	WEB SERVICE EMPLOYMENT	49
C.	OPERATING-SYSTEM INDEPENDENCE.....	50
D.	INTEROPERABILITY	52
E.	PARSER FUNCTIONALITY	53
F.	SUMMARY	53
VI.	CONCLUSIONS AND RECOMMENDATIONS.....	55
A.	CONCLUSIONS	55
B.	RECOMMENDATIONS FOR FUTURE WORK.....	57
1.	XML-Based Authentication and Authorization Technologies	57
2.	Web-Service Based Common Operating Picture (COP).....	58
3.	Web Service Discovery	59
	APPENDIX A – CLIENT CLASS.....	61
	APPENDIX B – CONNECTION CLASS.....	75
	APPENDIX C – SERVER CLASS.....	85
	APPENDIX D – PARSER WEB SERVICE CLASS	95
	APPENDIX E – TRANSFORMATION WEB SERVICE CLASS	101
	APPENDIX F – PHP CLIENTS.....	107
	BIBLIOGRAPHY	109
	INITIAL DISTRIBUTION LIST	113

LIST OF FIGURES

Figure 1.	This conceptual abstraction of a Network-Centric Warfighting construct connects common key warfighting elements, such as sensors, shooters and command and control, in order to allow better command and control and facilitate decision-making. (From Joint Publication 6.0: Doctrine for C4 Systems Support to Joint Operations, Figure II-4)	16
Figure 2.	The previous conceptual abstraction is easily translated into three sub-architectures: an Information Grid, a Sensor Grid and an Engagement Grid. This figure highlights the network-centric information flow between sensors, command and control and shooters. (From: http://www.dtic.mil/jcs/j6/education/warfare.html , September 2004)	17
Figure 3.	This example Operational Architecture for Network-Centric Warfare integrates a mission specific Sensor Grid and a mission specific engagement grid to enable Precision Engagement of Air Defense targets. (From: http://www.dtic.mil/jcs/j6/education/warfare.html , September 2004)	18
Figure 4.	The GIG ES provides the necessary services to bridge the Transport layer and the Application layer and consists of services such as electronic mail, application hosting and weapon-target pairing. (From: http://ges.dod.mil/about/solution.htm , September, 2004).....	19
Figure 5.	Net-Centric Enterprise Services (NCES) are the basis of GIG ES and consists of those core services that are relevant to all users. NCES will also contain extended service sets that are community specific, which are called Community of Interest (COI) services. (From: http://ges.dod.mil/about/solution.htm , September 2004).....	20
Figure 6.	A Grid based application designed to provide multiple services can output XML data from its internal data source as well as also implementing several user interfaces that output HTML, portlets or Personal Digital Assistant (PDA) data. (From: Assistant Secretary of Defense for Networks and Information Integration (ASD/NII), GES-CES-Strategy, Draft Version 1.1)	22
Figure 7.	Specialized services can extend existing services providing added value for all users and communities. (Assistant Secretary of Defense for Networks and Information Integration (ASD/NII), GES-CES-Strategy, Draft Version 1.1)	23
Figure 8.	The GIG architecture ensures users have access to any and all resources facilitating the transition from a “need-to know” methodology to a “need-to-share” methodology. (Network-Centric Operations and Warfare Reference Model, Draft Version 0.9)	24
Figure 9.	The Warfighter Information Network – Tactical (WIN-T) is a transport for the GIG. This figure provides a graphical representation of the	

	implementation from the WIN-T Operational Requirements Document (ORD). (Original source: Appendix k, Tab A to WIN-T ORD).....	25
Figure 10.	The FCS is a Joint networked system of systems, composed of eighteen subsystems, that uses the WIN-T platform as a backbone for its network infrastructure. (From presentation titled: “GIG from a Warfighter Perspective” authored by Wayne A. Van Dine, Jr., see Van Dine in the bibliography for more details)	26
Figure 11.	The target architecture for the proof of concept (POC) example depicted here is intended to demonstrate the flexibility and platform independent nature of Web services.....	29
Figure 12.	The “Locate File” dialog box is used to open a local file.....	31
Figure 13.	The “Current File” tabbed pane allows the user to view the selected comma separated file.	32
Figure 14.	The connection dialog is designed to allow the user to enter the Internet Protocol address and port number of the server.....	33
Figure 15.	The “Get File From Server” tabbed pane allows the user to select a file from the server.	34
Figure 16.	Retrieved data is viewed in the “Get File From Server” tabbed pane.	35
Figure 17.	The server graphical front-end is displayed when the server is running.	36
Figure 18.	The control statement used by the server determines which operation to perform based on the message received from the client.	37
Figure 19.	The server displays the Base-64 encoded file sent by the client.....	38
Figure 20.	The server “decode” method is exposed as a Web service and provides the basic services for retrieving data from the server.	39
Figure 21.	This screenshot shows Apache-SOAP running as a Web Application under Tomcat.	40
Figure 22.	The Apache SOAP Admin page is used to manage service deployment description configurations.....	40
Figure 23.	This screenshot shows the service deployment descriptor ID and methods list configuration form. These parameters are used to create the service deployment descriptor file used by SOAP.....	41
Figure 24.	The service deployment descriptor configuration form requires a Java Provider as well.....	41
Figure 25.	When the Web portal client is used to submit a request to the basic client, the results are displayed as text in a new browser window.	44
Figure 26.	The second service implemented parses the text and converts it into XML. The Firefox Web browser renders to resulting XML output from the parser service without the XML tree. Firefox implements XML parsing and rendering in a different manner than Internet Explorer.	45
Figure 27.	Although the results from the parser are the same as before, Internet Explorer renders the resulting XML output as an XML tree. This is specific to the Internet Explorer.....	46
Figure 28.	The last service implemented uses XSLT to transform XML into an HTML table containing a Summary Personnel Status Report as shown in Firefox.....	47

Figure 29.	As previously identified this sample deployment architecture for the proof of concept demonstrates the flexibility of Web services. The Java applications are deployable on Windows and Linux and the services function properly over a network.....	50
Figure 30.	Much effort was made to utilize open-source technologies during the conduct of this research. The Web server of choice for the proof of concept was Apache on both Windows and Linux. These statistics represent the market share for top servers across all domains from September 1995 to September 2004, which demonstrates the ubiquity of Apache. (From: http://news.netcraft.com/archives/web_server_survey.html , September 2004).	51
Figure 31.	Each button of the G1 Web portal calls a service designed to return specific data to demonstrate the capabilities of each service. Each service represents a stepping-stone of the final product.	52

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS, ACRONYMS AND SYMBOLS

API	Application Programming Interface
ASP	Active Server Pages
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance
COI	Community of Interest
COM/DCOM	Component Object Model/Distributed Component Object Model
COP	Common Operating Picture
CORBA	Common Object Request Broker Architecture
CSV	Comma Separated Value
DCE	Distributed Computing Environment
DOM	Document Object Model
EJB	Enterprise Java Beans
E-mail	Electronic Mail
ES	Enterprise Services
FCS	Future Combat Systems
GIG	Global Information Grid
HTML	Hypertext Markup Language
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries
JAX-RPC	Java API for XML Based RPC
JSF	Java Server Faces
JSTL	Java Server Pages Standard Tag Library
JWSDP	Java Web Services Developer Pack
MOL	Marine Online
NCES	Network Centric Enterprise Services
NCOW RM	Network Centric Operations and Warfare Reference Model

OASIS	Organization for the Advancement of Structured Information Standards
ORD	Operational Requirements Document
PDA	Personal Digital Assistant
PEAR	PHP Extension and Application Repository
PHP	PHP: Hypertext Preprocessor
POC	Proof of Concept
SAAJ	SOAP with Attachments API for Java
SIPRNET	Secret Internet Protocol Router Network
SME	Subject Matter Expert
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOM/DSOM	System Object Model/Distributed System Object Model
SOP	Standing Operating Procedure
TCP	Transmission Control Protocol
TO	Table of Organization
TPED	Task, Process, Exploit and Disseminate
TPPU	Task, Post, Process and Use
UD/MIPS	Unit Diary/Marine Integrated Personnel System
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
WIN-T	Warfighter Information Network - Tactical
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
X3D	Extensible 3D Graphics
X-KISS	XML Key Information Service Specification

XKMS	XML Key Management Specification
X-KRSS	XML Key Registration Service Specification
XML	Extensible Markup Language
XMLP	XML Protocol

THIS PAGE INTENTINALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge and give thanks to the following:

Cryptographic Research Laboratory manager Nathan Beltz; Nathan consistently provided the necessary equipment and support throughout the course of the research.

Applied Technology Division, National Security Agency for funding the research.

Research Associate Curtis Blais; Curt's enthusiasm was always uplifting and infectious. He became the sounding board for many of the author's rantings, always doing his very best to provide time to support his students.

I would like to give a very special thanks to my wonderful Wife, Melissa and our three beautiful children, Brandon, Hunter and Cody for their unwavering support and love.

And finally, I would like to thank our Lord and Savior, Jesus Christ. Through Jesus, all things are possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

Currently under development, the Global Information Grid (GIG) Enterprise Services (ES) is a suite of capabilities intended to provide improved user access to mission-critical data via Web-based and network technologies. Some of the problems of implementing such capabilities include non-uniform data formats, incompatible run-time environments and nonstandard proprietary applications, all of which block operational interoperability.

Web services are specifically designed to address the interoperability challenges of a service-oriented architecture (SOA) such as the GIG. SOAs are networked infrastructures that are designed to facilitate the interoperability of collections of services without requiring service context awareness. Standards-based Web services provide the necessary flexibility and extensibility to ensure information flow is platform, run-time and software independent.

The proof of concept (POC) software example developed for this research demonstrates the flexibility and extensibility of standards-based, operating-system-independent Web services. The result is an experimental endeavor to provide a mock operation command center information portal, which provides a notional summary personnel status report to the commander in real-time from a Web service that was originally generated by a stand-alone client/server system. The POC is developed with great attention to open-source technologies and open-standards compliance. The key technologies involved are Extensible Markup Language (XML), the Java programming language, PHP: Hypertext Preprocessor (PHP)¹ scripting language and Simple Object Access Protocol (SOAP).

This work demonstrates the benefits of leveraging Web services to unlock legacy specialized applications to enhance the Warfighter's battlespace awareness by improving information flow via a Web based information portal.

¹ PHP is considered a recursive acronym. The acronym "PHP" is itself a part of the literal name. See <http://www.php.net/manual/en/introduction.php> for more information.

B. PROBLEM SPACE

Timeliness of information has always been key to a commander's critical decision-making. The systems currently used to assist in that process are unable to provide a comprehensive perspective of the battlespace in real-time, nor does the current architecture support proper and necessary staff integration, which requires the sharing and accessibility of information across the entire staff.² This research addresses some of the overarching shortcomings of proprietary, closed source systems that can ultimately require contractor lock-in or in some cases just become unsupportable.

This research assumes that one priority of technological improvement is to aid humans with tasks that otherwise require greater effort, manpower and in many cases expertise. For the purposes of this research the author focuses on information management capabilities and limitations. Today's computer systems are extremely efficient at processing massive amounts of data. The improvements in hardware, software and network technology allow for storage, management and access of information from many local and remote resources simultaneously. However, the manner in which these information management assets are implemented is the source of the problem this research addresses. This paper examines corporate "best practices" and draws conclusions pertaining to which implementations can benefit military organizations.

The resulting POC is based on standards specified under the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS). Specifically, the Web service architecture is based on requirements set forth by the Web Services Architecture Working Group.

For the most part, current practices within the Marine Corps require several software applications to gather and correlate relevant data. For example, within Marine Corps Personnel and Administration the system used to maintain personnel data is the Unit Diary/Marine Integrated Personnel System (UD/MIPS)³. The data is maintained on an IBM 3270 mainframe in a proprietary format such that even though commanders have

² This statement is an opinion based on the author's experience as the I MEF Personnel Officer and is the premise for this research.

³ For more information on UD/MIPS see: <http://www.missa.manpower.usmc.mil>

access via a closed source client, it provides virtually no ability to integrate that data with any other report. If a commander needs to create a report utilizing the data in UD/MIPS it is necessary to manually extract the data and import into another application such as Microsoft Excel. If the report is needed by “higher” then it is reformatted, if necessary, to their specification and forwarded via email or formal message.

The POC Web service based architecture is also used to explore how to extend legacy systems and improve data integration via a network without needing to completely re-architect the existing infrastructure.

Other assumptions are identified and qualified within the conclusions drawn from this research in Chapter VI.

C. MOTIVATION

The motivation behind this thesis is to improve the commander’s ability to quickly and efficiently utilize all of the systems at his or her disposal, locally or remotely, to see any and all data relevant to the battlespace as close to real-time as practicable. The research explores how Web services may be used to enhance our capabilities to migrate and integrate data from legacy and proprietary systems virtually in real-time into an environment such as a web-based reporting portal.

1. Personal Background

The following is provided to establish some context for the information within this document. This also addresses the author’s personal motivation behind this research.

The author’s first duty assignment coincided with a significant reduction in manpower within the Personnel and Administration field, which led to consolidation of administration assets and a very quick push to develop software tools to automate many administrative requirements. This was typically driven by a higher echelon command and the results were not necessarily beneficial to lower-level commands.⁴

As a result of experiences from his first assignment the author developed a significant sensitivity to the manpower issues of the subordinate commands and elements. Therefore, upon reassignment to a higher command, he made every effort to lighten their

⁴ Upon commissioning the author was assigned to 1st Maintenance Battalion, 1st Force Service Support Group (FSSG) as Adjutant for about one and a half years and subsequently assigned to First Marine Expeditionary Force Command Element (I MEF CE) as Personnel Officer for about two and a half years.

burden whenever possible. The tools that were available were definitely more capable but were still often inadequate and required a great deal of manual intervention to accommodate the needs. For example, there have been great strides made toward improving access to, and use of, the data maintained by the personnel reporting system UD/MIPS in the form of Marine Online (MOL)⁵. However, the information is still not easily utilized to accommodate many of the reporting requirements an administrator is responsible for.

Although some of the former commentary has pertained to a particular system and its limitations, this research does not demonstrate how any specific system can be improved by implementing Web services, rather it demonstrates how, conceptually, any application can be extended to provide more robust and flexible services.

2. The Need for Improved Integration and Extensibility

Within any military organization there are many reporting requirements, whether peacetime or wartime, which have to be processed in order to maintain a necessary state of readiness. Some of these requirements have routine submission deadlines but many do not. Typically the information submitted is gathered via proprietary client/server applications that are networked together as in the UD/MIPS example above.

The military faces numerous constraints and limitations as a result of systems and applications that are not extensible and do not integrate. This is obvious across all sections of a military staff. Today, when a commander is briefed, the data provided is usually based on information gathered within a given time period, which can often be several hours old. The information comes to the command center via many methods. Personnel reporting is typically done via electronic mail (E-mail) with spreadsheets. Some Intel systems provide data in a more timely manner but still require subject matter experts (SMEs) to extract relevant data and manually update other reports to provide the commander, as well as higher, with an accurate common operating picture (COP).

The scope of this research involves answering the following:

⁵ Marine Online (MOL) at <https://tfas.mol.usmc.mil/TFAS/login.do> is a personnel administration web site designed to allow individual Marines access to their personnel information.

- What type of data is needed for the POC? Details are provided in Chapter IV.
- How operating-system independent are Web services? This is demonstrated through the POC and is discussed in Chapter IV.
- What are the keys to Web service interoperability? This is addressed in the technical assessment portion of Chapter II.
- What are the keys to successful implementation in a military operational environment? This is discussed in Chapters V and VI.
- What software is required? This is discussed in chapter IV.
- What mechanisms can be implemented to aid in addressing security issues such as authentication and authorization? This is addressed in Chapter VI.

3. The Need for Web-based Interoperable Solutions

The significance of a Web-based interoperable solution is that it provides the flexibility necessary to build an architecture without a “single point of failure.” It also utilizes standards that have been tested in a dynamic corporate environment, which provides the benefit of lessons learned. The Web provides redundancy and global reachability.

This approach also provides scalability, allowing for implementation on smaller networks that are bridged and secure as well as large networks that have complete access to the Internet or Secret Internet Protocol Router Network (SIPRNET) for secure operations.

D. THESIS ORGANIZATION

Chapter II provides background information pertaining to the technologies and protocols researched and employed in the POC as well as a brief technical assessment of the status of Web service “best practices.”

Chapter III is devoted to explaining the concepts behind the GIG, Grid computing and Network-Centric Warfare.

Chapter IV details the development of the POC. The functionality of each component is explored in depth. The goal of this chapter is to ensure a proper understanding of each component’s role in the overall architecture.

Chapter V discusses the results of the research and the POC. The development process yielded lessons learned about design considerations, interoperability issues and efficiency concerns.

II. RELATED WORK

A. INTRODUCTION

This chapter covers technical details and background of technologies utilized in development of the POC. Since the POC involves Java, SOAP, and XML, W3C and the Web Services Interoperability (WS-I) organization as well as Sun Java are discussed below. Although not implemented in the POC, Universal Description, Discovery and Integration (UDDI) is briefly discussed in order to provide some background information for possible future work.

Finally, this chapter concludes with a brief technical assessment of the technology behind today's Web service implementations. The technical assessment addresses three main areas of concern: what is a Web service, how is a Web service used and how can it improve the operational picture for the commander.

B. CURRENT OPEN SOURCE WEB SERVICE STANDARDS ORGANIZATIONS

1. World Wide Web Consortium (W3C)

The W3C is the standards organization that maintains and contributes work to standards significant to interoperable communication over the World Wide Web. W3C has been crucial in the development and maintenance of many key protocols and specifications such as Hypertext Markup Language (HTML), Document Object Model (DOM), SOAP, XML and over 50 other recommendations.

The W3C organization is comprised of several Activities, each of which maintains its own structure typically consisting of multiple Working Groups, Interest Groups and Coordination Groups. It is from within these activities that the groups usually produce recommendations and technical reports. Of specific interest to this thesis is the work contributed by the Web Services Activity. All work pertaining to Web services is managed as part of W3C's Architecture Domain. The Web Services Activity structure consists of three Working Groups and an Interest Group, which are all coordinated by a single Coordination Group: (W3C, September 2004)

- XML Protocol Working Group
- Web Services Description Working Group
- Web Services Choreography Working Group
- Semantic Web Services Interest Group

The XML Protocol Working Group maintains several documents relevant to this thesis. Specifically relevant are the XML Protocol (XMLP) Requirements working draft document and SOAP specification. Development of the POC involves the use of SOAP and XML fairly extensively. XML is the fundamental language used as the basis for the extensibility of Web services and the implementation of SOAP. SOAP is the protocol used to communicate between SOAP processors over the network commonly bound to such protocols as HTTP and XML-RPC over Transmission Control Protocol (TCP). A SOAP message consists of several pieces. Generally speaking there is the SOAP Envelope, which is the outer most element, the SOAP Header and SOAP Body. (Web Services Description Working Group, 3 August 2004)

The Web Services Description Working Group maintains the working draft of the Web Services Description Language (WSDL) specification. WSDL is designed to provide an XML language for describing Web services. Typically this is used to document the details of how a service requester can properly interface with a service provider and is commonly included with the UDDI registration of a specific service. (UDDI, September 2004)

The Web Services Choreography Working Group maintains the working draft of the Web Services Choreography Description Language (WS-CDL). WS-CDL is an XML-based language that is designed to facilitate peer-to-peer collaborations of Web services participants by defining their common and complementary observable behavior. The end result ensures ordered message exchanges accomplish a common business goal.

Although these are the only working groups from W3C discussed here, there are many other Activities within the W3C that provide critical technology to the development

and advancement of Web services, such as the Semantic Web Activity, the XForms Activity, the XML Encryption Activity and the XML Activity just to name a few. (W3C, September 2004)

2. Web Services Interoperability (WS-I) Organization

The WS-I organization is an industry effort intent on promoting Web services interoperability across platforms and programming languages. WS-I provides a community of members consisting of software vendors of all sizes, enterprise customers and essentially any others interested in furthering Web services interoperability. (WS-I, September 2004)

As an example of the criticality of the WS-I Basic Profile we can look at two separate but common Web service development platforms, .NET and Java. Prior to the WS-I Basic Profile, services developed under .NET yielded implementations that were not necessarily properly accessible from a requester developed in Java due to the nonstandard manner in which service request processing occurred. With the WS-I Basic Profile and current tools contributed by WS-I, it is possible to ensure a specific level of conformance and hence improved interoperability. The WS-I Basic Profile does not provide a guarantee of total interoperability but it does address the most common problems that implementation experience has revealed to date. (Ballinger, Keith et al, 24 August 2004)

3. Organization for the Advancement of Structured Information Standards (OASIS)

OASIS is fundamentally focused on electronic business (e-business) advancement through well defined worldwide standards for security, Web services, conformance, business transactions, supply chain, public sector and interoperability within and between the marketplace. (OASIS, September 2004)

OASIS is responsible for many specifications that are significant to the overall development of a total Enterprise-level Web service solution, for the purposes of this research the primary standard of interest is the UDDI specification. UDDI is a collection of searchable web-based registries that allows entities to publish information about themselves. The UDDI registry contains Yellow Pages, White Pages and Green Pages. Yellow Pages contain registration classification information about the business entities or

their services under different categories. White Pages contain listings of the existing business entities. Green Pages provide technical information on how to invoke a certain service. The end result is a searchable registry that allows service consumers the ability to locate specific services and requirements pertaining to their employment. There are many proprietary UDDI registry solutions available on the market today. Due to the work of the OASIS UDDI Specification Technical Committee there is a single standard that most UDDI registries try to abide by. As of this writing the UDDI version 2 specification has been promoted as an OASIS standard with version 3 in revision. (UDDI, September 2004)

C. JAVA-BASED WEB SERVICES

There are two significant reasons Java was chosen as the basis for development of the POC example in this thesis:

1. It has a well supported and mature Application Programming Interface (API) for developing network programs, and the latest release of the Java Web Services Developer Pack (JWSDP 1.3) is a complete Web services development platform.
2. Java, although not open source as of this research, is a freely distributed API and run-time environment and runs on all OSs of interest.

JWSDP 1.3 needs further description as this is considered one of the most significant toolkits available for developing Web services due to its comprehensive nature. The JWSDP consists of the following: (JWSDP, September 2004)

- Java Server Faces (JSF) v1.0 EA4 – a standard specification for building User Interfaces (UI) for server-side applications.
- XML and Web Services Security v1.0 EA2 – used to secure SOAP messages.
- Java Architecture for XML Binding (JAXB) v1.0.2 FCS – binds XML schemas to Java representations.
- Java API for XML Processing (JAXP) v1.2.4 FCS – enables applications to parse and transform XML documents.
- Java API for XML Registries (JAXR) v1.0.5 FCS – provides a uniform and standard Java API for accessing different kinds of XML registries.

- Java API for XML-based Remote Procedure Calls (JAX-RPC) v1.1 FCS – provides XML-based RPC capability to Web applications.
- SOAP with Attachments API for Java (SAAJ) v1.2 FCS – provides a standard method of sending XML documents over the Internet from a Java platform.
- Java Server Pages (JSP) Standard Tag Library (JSTL) v1.1 EA – a tag library designed to encapsulate many of the common core functionalities found in Web applications.
- Java WSDP Registry Server v1.0_06 FCS – a Java-based UDDI registry implementation.
- Ant Build Tool 1.5.4 FCS – a Java based build tool.
- Apache Tomcat v5.0 EA development container – Tomcat is a Java servlet container that allows for the deployment of Java based applications as services on a network.

JWSDP has been tested on the following OSs with Java 2 Standard Developer Kit, Standard Edition, versions 1.4.1_xx and 1.4.2_xx:

- Sun Solaris Operating Environment 8 and 9, and Sun Solaris 9 for X86
- Windows 2000 Professional Edition
- Windows XP Professional Edition
- RedHat Linux 8.0

Much has been contributed to JWSDP from open source organizations such as the Apache Software Foundation (ASF)⁶, which maintains active development of Ant and Tomcat.

D. BRIEF TECHNICAL ASSESSMENT OF WEB SERVICES

1. Problem Resolution Through Maturing Standards and Technology

As XML and Web services as a whole mature, their uses are becoming increasingly prevalent throughout the Internet. The extensibility and flexibility of Web service implementation is well suited to deployment in today's global, internet-driven

⁶ More information on the Apache Software Foundation can be found at <http://apache.org>

business commerce and government service provisioning. In the past ten years or so, many distributed technologies such as Distributed Computing Environment (DCE) (The Open Group, September 2004), System Object Model/Distributed System Object Model (SOM/DSOM) (Webopedia, September 2004), Common Object Request Broker Architecture (CORBA) (Object Management Group, September 2004), Enterprise JavaBeans (EJBs) (J2EE, Sun Java, September 2004), Component Object Model/Distributed Component Object Model (COM/DCOM) (Microsoft, September 2004) and others, have been developed to address specific interoperability needs on the Internet. However, Web services provide organizations the ability to address such a wide range of needs that its use is quickly changing the way business is being done across the Internet, which is promoting standards development for its component protocols.

2. Defining Web Services

So, what is a Web service? Until recent years this question has led to many arguments among technical experts. Some argued that any technology that utilized the Web to provide a service is a Web service. This description included such architectures as database driven Web sites that deliver dynamic content on demand utilizing such technologies as Microsoft's Active Server Pages (ASP) or the widely used PHP scripting language, both of which can be embedded within HTML when creating Web sites. There are many other technologies, which are considered to provide Web services, that utilize protocols such as Transmission Control Protocol/Internet Protocol (TCP/IP) that drive today's Internet. For the purposes of this research, a Web service is defined by the W3C's definition:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (Haas, 11 February 2004)

3. Best Practices

It is not uncommon for new technologies to have implementations rushed to market without a mature, agreed-upon standard method of implementation. As an example, consider two typically competing technologies that are common platforms for

implementing Web services: Sun's Java and Microsoft's .NET architecture. Even though Web services are supposed to support interoperability, each of these development platforms has a unique manner of effecting Web service implementations, so that a .NET service requester is not consistently able to communicate with a Java-based service provider. This type of issue, along with many others, was the motivation for large industry participants organizing the WS-I organization, which released the final WS-I Basic Profile 1.1 on August 24, 2004 (Ballinger, 24 August 2004). Hopefully this and related progress will lead to interoperable best practices that align with W3C recommendations.

4. Extensibility is Crucial

Even with XML's extensibility, Web services are not likely to solve every business process problem. There are many issues and concerns about Web services deployment that still need to be resolved. Some are a part of the natural evolution of an innovation but others are a part of the basic nature of technology. The maturity of the technology has to keep up with rapidly changing Internet infrastructure. However, because of its flexibility, it is likely to evolve and sustain better than most other technologies attempting to provide similar capabilities. WS-I is likely to play a key role in extending Web services beyond inter-organizational use to individual users as well.

5. Web Service Usage

Considering the flexibility associated with a Web service architecture, implementation could be very extensive. In an effort to provide scope for the POC this research implements Web services in such a manner as to integrate data from different sources into a single reporting mechanism. These different sources, or applications, mimic systems that are otherwise unable to communicate with each other and therefore are not able to share data. Web services operate across different OSs utilizing an intranet to demonstrate interoperability and flexibility. The POC also consists of Web services that are of different technologies. Web services facilitate access to data created and/or maintained by a Java application. More detail is given in chapter IV. The final POC provides a seamless integration of data quickly allowing the commander to access a real-time information summary.

E. SUMMARY

This chapter provided background information pertaining to the technologies and standards utilized during the development of the POC and presented a brief technical assessment of Web services.

III. THE GLOBAL INFORMATION GRID (GIG)

A. INTRODUCTION

The concept of Grid computing provides the ability to combine the computing power of many networked computers. This concept is being realized in the form of the Department of Defense's (DoD) GIG initiative to facilitate the needs of Net-Centric Warfare.

B. THE CONCEPT OF GRID COMPUTING

Grid computing is the progression of networked computers into a seamlessly integrated, virtual computing space with shared resources providing unprecedented computing power, services and information with incredible collaborative capability. Although the Internet has provided a means to allow computers to communicate, the concept of Grid computing enables computers to work together. One of the primary tenets of Grid computing is that it is capable of connecting heterogeneous computing platforms and data sources such that they appear to the end-user as a single computing system and data store.

One application of Grid computing implements software that allows networked computers to be used by researchers to process enormous volumes of data. The software can potentially coordinate the processing of thousands of personal computers; each doing a small part, that when combined provides enormous computational power. The computational power realized is often faster and more efficient than monolithic supercomputers (Walker et al 2004, 3).

The previous example was a limited representation of Grid computing and does not demonstrate the full capacity of available capabilities. The concept of the GIG better represents the potential of Grid computing with the application of Network-Centric Warfare and Network-Centric Information Operations on a Grid computing architecture. Grid computing provides the performance, capacity and adaptability that is required to advance Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) systems integration in order to better task, process, exploit and disseminate (TPED) the enormous amounts of data provided from modern sensors

and open sources (Walker et al. 2004, 4). Fully implementing these capabilities in the GIG architecture also facilitates the transition from the TPED process to a task, post, process and use (TPPU) paradigm, which is discussed in greater detail later. (“The Solution”, September 2004)

C. A JOINT WARFIGHTING PERSPECTIVE ON THE CONCEPT OF NETWORK-CENTRIC WARFARE

Network-Centric Warfare is an implementation of Grid computing specifically focusing on linking command and control, sensors and shooters together to increase Joint combat power. Network-Centric Warfare has emerged as a result of the development of programming languages, protocols and standards that allow for platform independent communication such as, HTTP, XML and Java. Figure 1 shows a conceptual abstraction of a Network-Centric Warfighting construct connecting common key warfighting elements, such as sensors, shooters and command and control, in order to allow better command and control and facilitate decision-making.

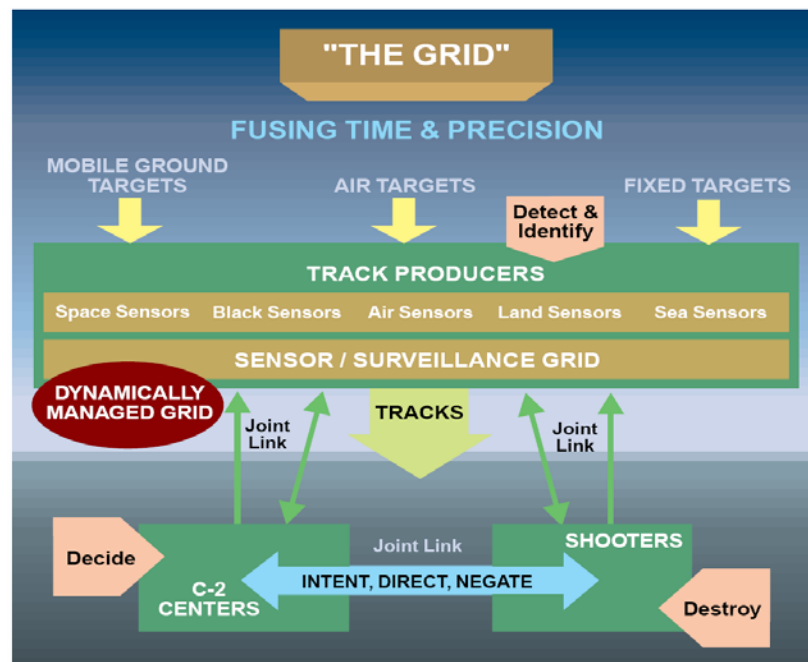


Figure 1. This conceptual abstraction of a Network-Centric Warfighting construct connects common key warfighting elements, such as sensors, shooters and command and control, in order to allow better command and control and facilitate decision-making. (From Joint Publication 6.0: Doctrine for C4 Systems Support to Joint Operations, Figure II-4)

As shown in Figure 2, the previous conceptual abstraction is easily translated into three sub-architectures: an Information Grid, a Sensor Grid and an Engagement Grid. The Information Grid is the fundamental building block consisting of both military and commercial communication capabilities providing the necessary network infrastructure to ensure proper resource connectivity and enables the Sensor Grid to generate battlespace awareness. The Sensor Grid consists of the assets necessary to provide the Joint Force Commander the necessary awareness across the Joint battlespace. These assets are represented by sensor peripherals and sensor applications. Sensor peripherals consist of space, air, ground, sea and cyberspace based sensors as well as embedded sensors that facilitate tracking levels of consumables such as fuel and munitions. The Sensor Grid applications are those applications required by the sensor platforms. The Engagement Grid takes advantage of the benefits of the improved battlespace awareness provided by the Sensor Grid and allows the Joint Forces Commander the ability to maximize the employment of forces.

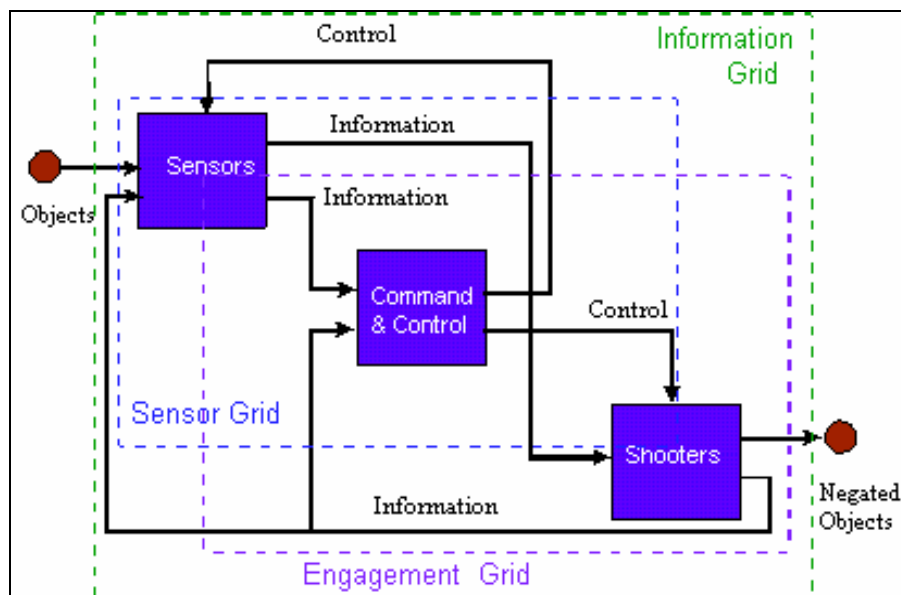


Figure 2. The previous conceptual abstraction is easily translated into three sub-architectures: an Information Grid, a Sensor Grid and an Engagement Grid. This figure highlights the network-centric information flow between sensors, command and control and shooters. (From: <http://www.dtic.mil/jcs/j6/education/warfare.html>, September 2004)

Figure 3 represents an emerging operational architecture for Network-Centric Warfare representing the integration of a mission specific Sensor Grid and a mission specific Engagement Grid. The employment of this operational architecture increases Joint combat power and effectiveness as a result of a combination of the Sensor Grid's ability to provide increased battlespace awareness and the Engagement Grid's capability to exploit it. (Net-Centric Warfare, September 2004)

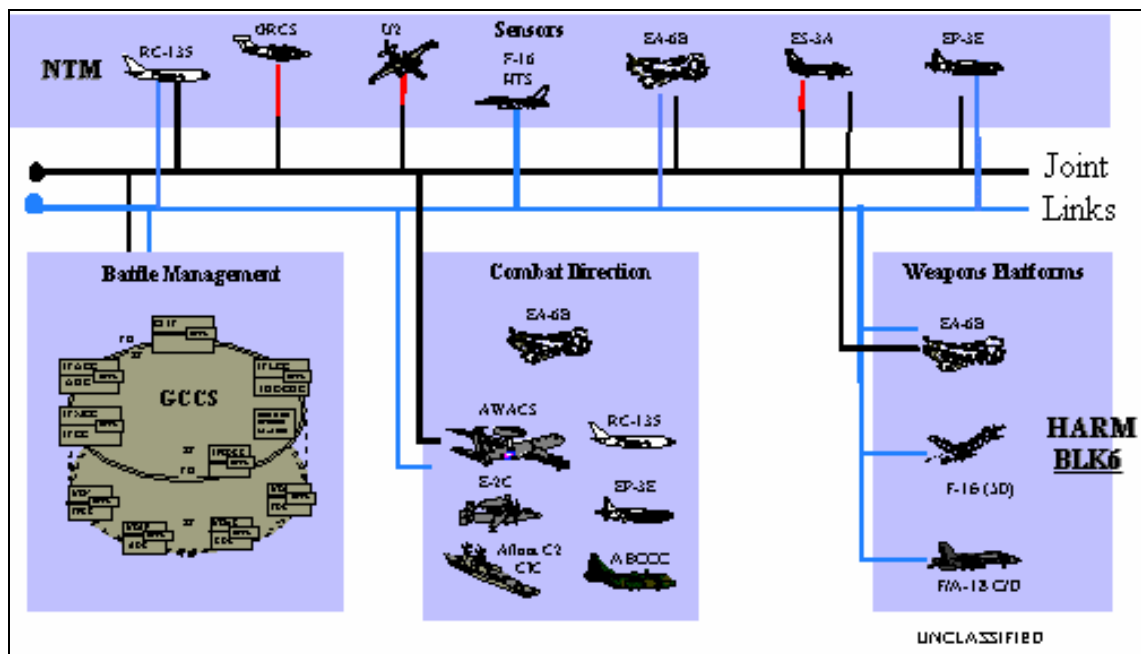


Figure 3. This example Operational Architecture for Network-Centric Warfare integrates a mission specific Sensor Grid and a mission specific engagement grid to enable Precision Engagement of Air Defense targets. (From: <http://www.dtic.mil/jcs/j6/education/warfare.html>, September 2004)

D. GIG COMPOSITION

The GIG is designed to apply the concepts of Network-Centric Warfare to a Grid computing architecture. The GIG provides more capability than just increased battlespace awareness. Figure 4 shows the GIG architecture consisting of a Domain layer, Application layer, GIG ES layer, Transport layer and Management layer. The Management layer consists of such things as doctrine, governance, policy, standards, architecture and engineering, which establish the business processes and guide implementation. The Transport layer encompasses the physical infrastructure consisting

of such systems as the Defense Information Systems Network, Joint Tactical Radio System and Transformational Communications System. The GIG ES is designed to simplify resource access to eligible users facilitating information and decision superiority in every situation. The GIG ES provides the necessary services to bridge the Transport layer and the Application layer and consists of services such as electronic mail, application hosting and weapon-target pairing. The Application layer consists of those user applications necessary to acquire, process and use information and services such as the Deployable Joint C2 Program and the Business Management Modernization Program. Finally, at the top is the Domain layer, which determines the scope of deployment and usage of services. (“The Solution”, September 2004)

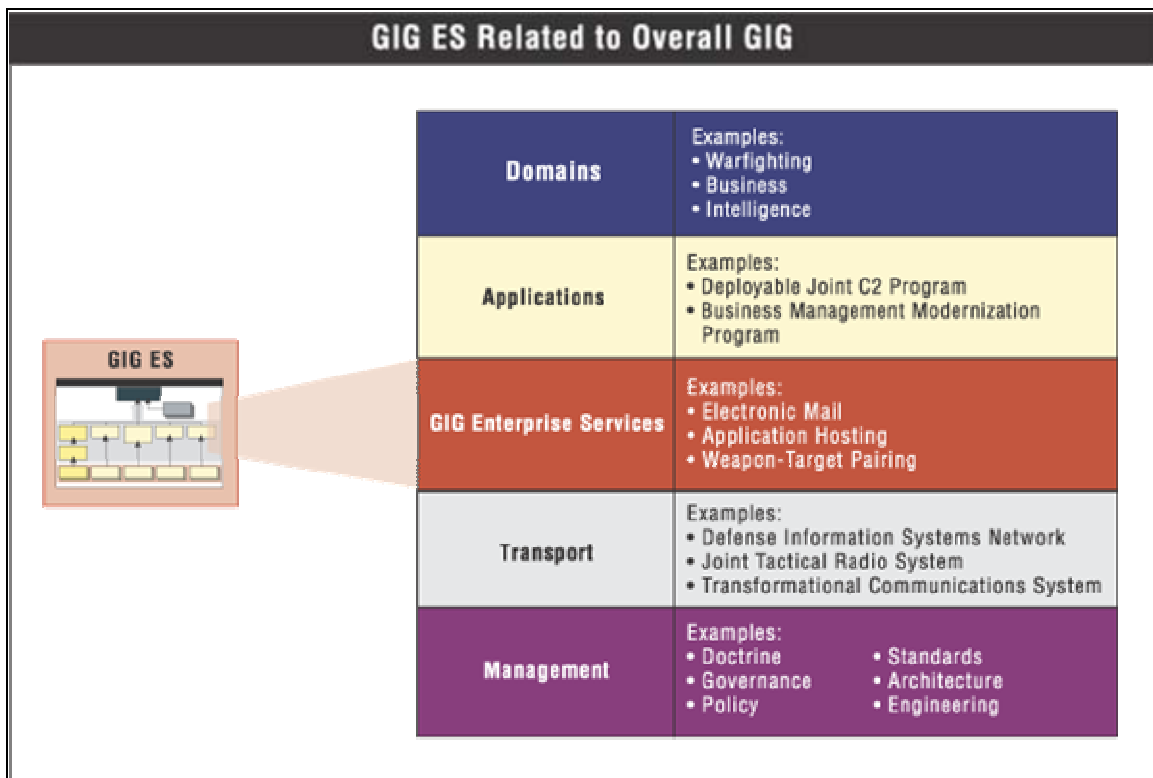


Figure 4. The GIG ES provides the necessary services to bridge the Transport layer and the Application layer and consists of services such as electronic mail, application hosting and weapon-target pairing. (From: <http://ges.dod.mil/about/solution.htm>, September, 2004)

1. Global Information Grid Enterprise Services (GIG ES)

The DoD is transitioning from a centralized TPED paradigm of planning to a net-centric TPPU paradigm of planning, employing information technology and connectivity to allow consumers to “pull” the information necessary to accomplish their mission. This is a significant shift from the previous methodology, which required the consumer to specifically identify its needs to the producer and depend on the producer to accommodate those needs. GIG ES is designed to facilitate that transition by providing the services necessary to meet the information needs of every authorized user across the DoD. Currently the focus of development is on those core services, referred to as Net-Centric Enterprise Services (NCES) that are relevant to all users. NCES will also contain extended service sets that are community specific, which are called Community of Interest (COI) services. Figure 5 shows a basic representation of NCES under the GIG ES concept of employment. (“The Solution”, September 2004)

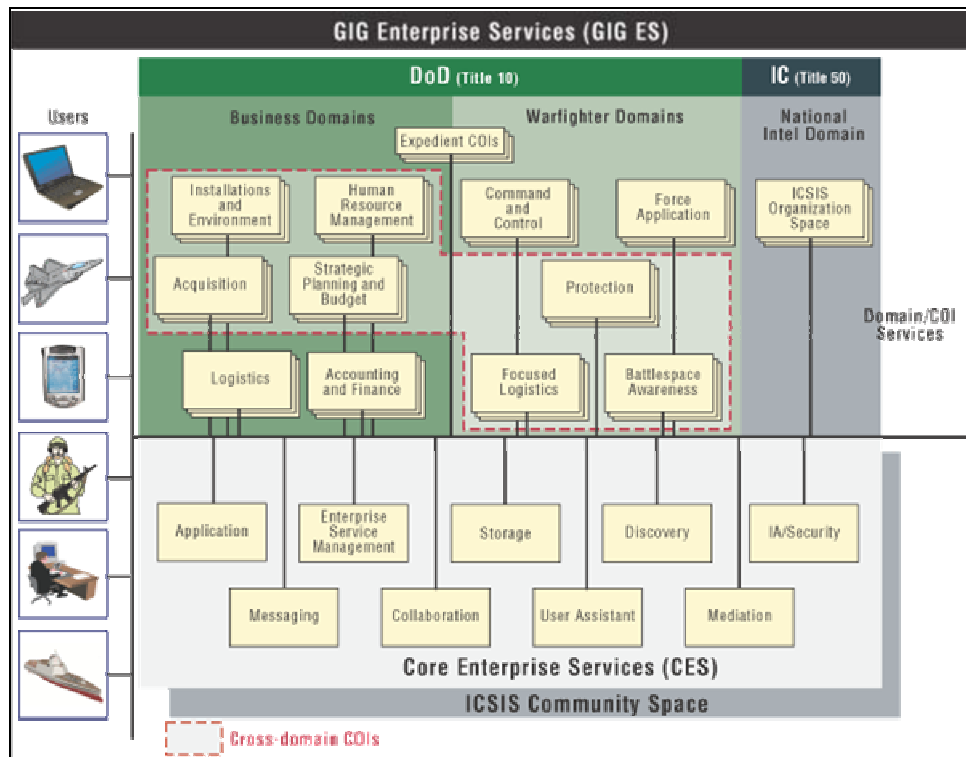


Figure 5. Net-Centric Enterprise Services (NCES) are the basis of GIG ES and consists of those core services that are relevant to all users. NCES will also contain extended service sets that are community specific, which are called Community of Interest (COI) services. (From:

<http://ges.dod.mil/about/solution.htm>, September 2004)

The following defines NCES that are currently under development for the GIG ES: (Meyerriecks 2004, <http://ges.dod.mil/articles/netcentric.htm>)

Enterprise Management Services – This set of services provides end-to-end GIG performance monitoring, configuration management and problem detection/resolution, as well as enterprise IT resource accounting and addressing, for example, for users, systems and devices. Additionally, general help desk and emergency support to users is encompassed by this service area, similar to 911 and 411.

Messaging Services – This service grouping provides the ability to exchange information among users or applications on the enterprise infrastructure, such as e-mail, DoD-unique message formats, message-oriented middleware, instant messaging and alerts.

Discovery Services – This service provides processes for discovery of information content or services that exploit metadata descriptions of IT resources stored in directories, registries and catalogs (to include search engines).

Mediation Services – This set of services helps broker, translate, aggregate, fuse or integrate data.

Collaboration Services – These services allow users to work together and jointly use selected capabilities on the network—for example, chat, online meetings and work group software.

Storage Services – These services provide physical and virtual places to host data on the network with varying degrees of persistence, such as archiving, continuity of operations and content staging.

Security Services – This set of services provides capabilities that address vulnerabilities in networks, infrastructure services or systems. Further, these provide characterizations of the “risk strength” of components as well as “risk posture” of the hosting run-time environment in support of future dynamically composed operational threads.

User Assistance Services – These services are automated “helper” capabilities that reduce the effort required to perform manpower intensive tasks.

2. Multiple Services Provided by a Single System

One of the goals of GIG ES is to provide an environment that allows for simple development and deployment of services as the need grows. What is traditionally

thought of today as a single system or application can simultaneously support multiple services under a properly implemented Grid architecture. As an example, a single data source provider that is designed to output XML data from its internal data source could also implement several user interfaces that output HTML, portlets or Personal Digital Assistant (PDA) data as represented in Figure 6. (“Core Enterprise Services Strategy – Draft Version 1.1”, 1 July 2003)

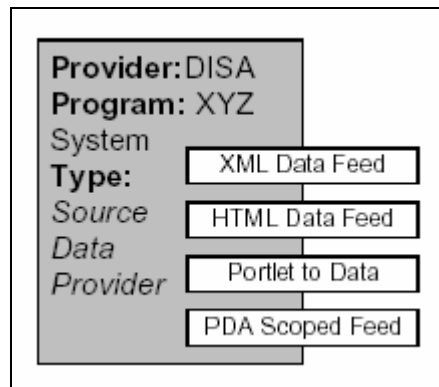


Figure 6. A Grid based application designed to provide multiple services can output XML data from its internal data source as well as also implementing several user interfaces that output HTML, portlets or Personal Digital Assistant (PDA) data. (From: Assistant Secretary of Defense for Networks and Information Integration (ASD/NII), GES-CES-Strategy, Draft Version 1.1)

A significant benefit of the extensible nature of the GIG is the ability to facilitate specialization. Under the traditional architecture this has often led to problems with interoperability as specialized applications are developed without extensible or flexible capabilities to interface with other applications or systems. Specialization under the GIG architecture allows for value added services to be realized from existing data providers as DoD organizations constitute the existing content into new specialized services for the rest of the community to use as represented in Figure 7. (“Core Enterprise Services Strategy – Draft Version 1.1”, 1 July 2003)

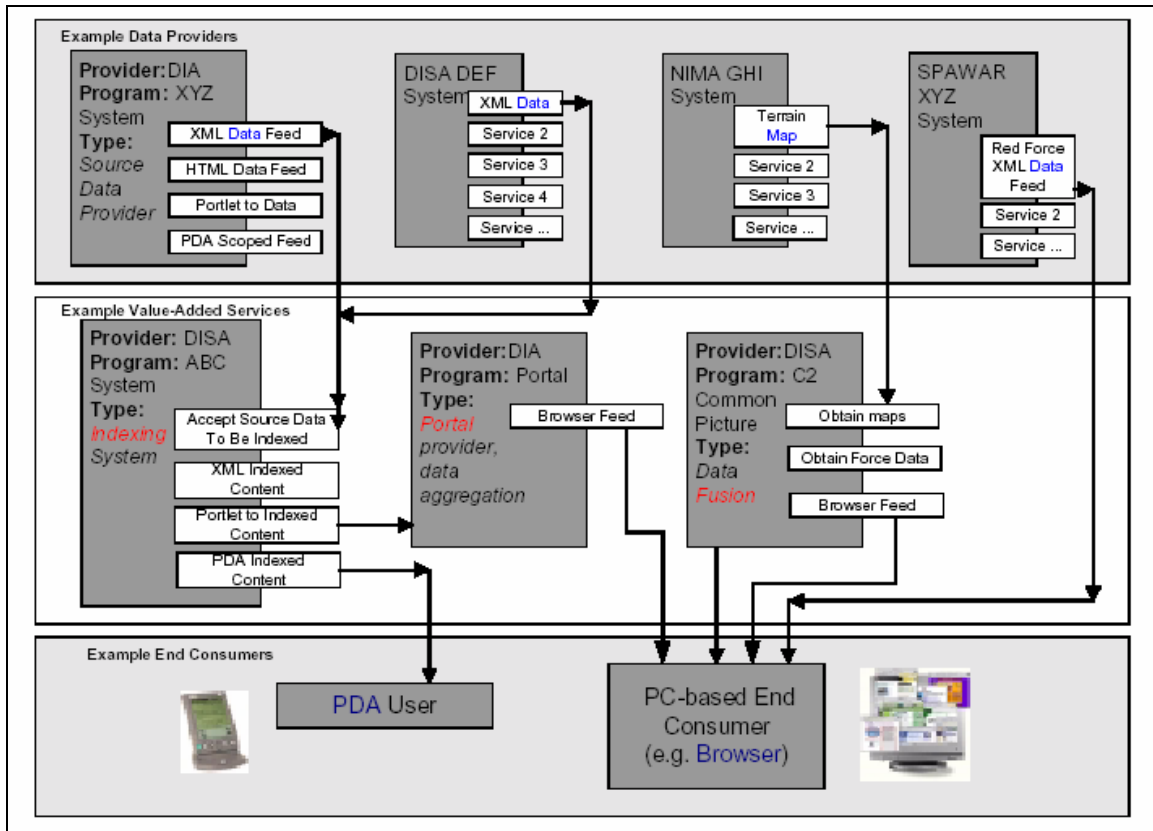


Figure 7. Specialized services can extend existing services providing added value for all users and communities. (Assistant Secretary of Defense for Networks and Information Integration (ASD/NII), GES-CES-Strategy, Draft Version 1.1)

E. THE WARFIGHTER'S PERSPECTIVE

The purpose of the following information is to provide the "Operational View" of the Network Centric Operations & Warfare (NCOW) Reference Model (RM) to better reflect the actual warfighter's perspective as identified by programs such as Warfighter Information Network – Tactical (WIN-T), Future Combat Systems (FCS), Land Warrior, GIG ES, etc.

As Figure 8 shows, employment of this architecture ensures users have access to any and all resources facilitating the transition from a “need-to-know” methodology to a “need-to-share” methodology. This presents another issue, however. Enforcing the “need-to-share” methodology does not imply that everyone has a need-to-know, so mechanisms must be imposed to ensure proper access controls are in place. (Van Dine, 27 October 2003)

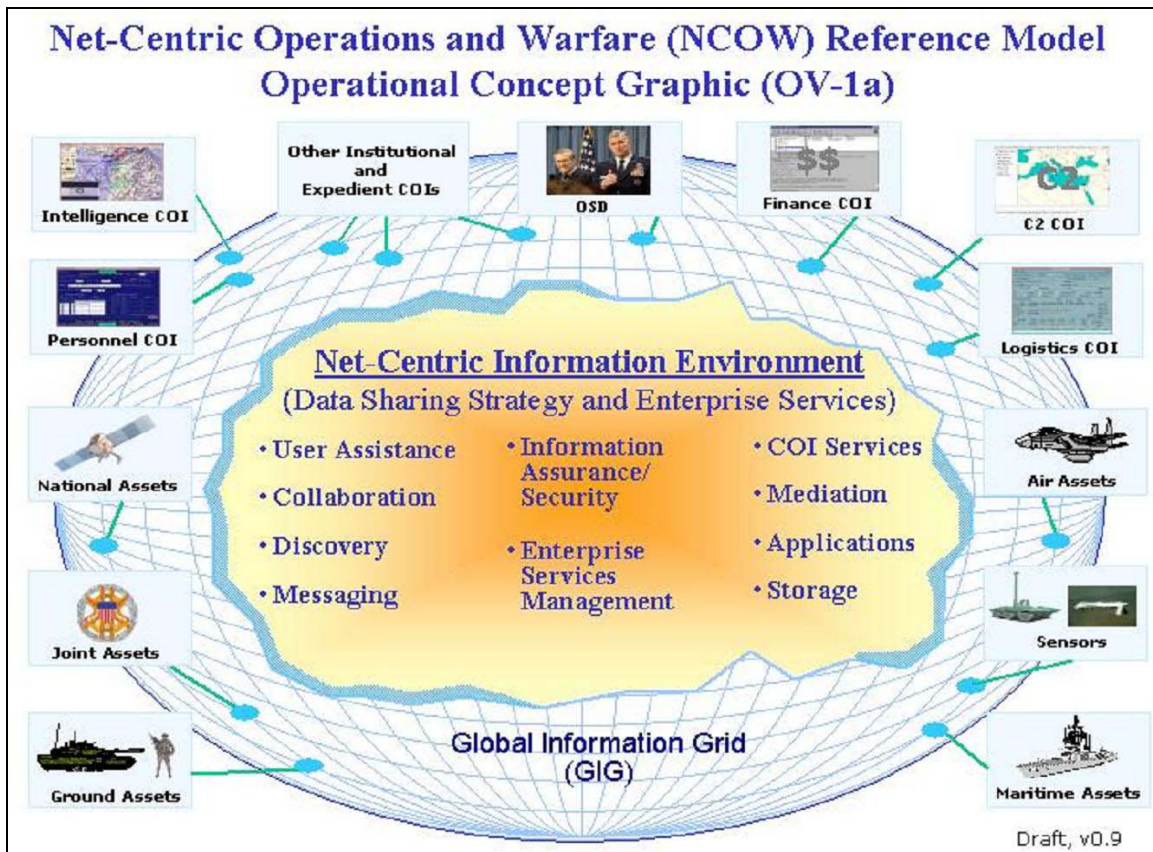
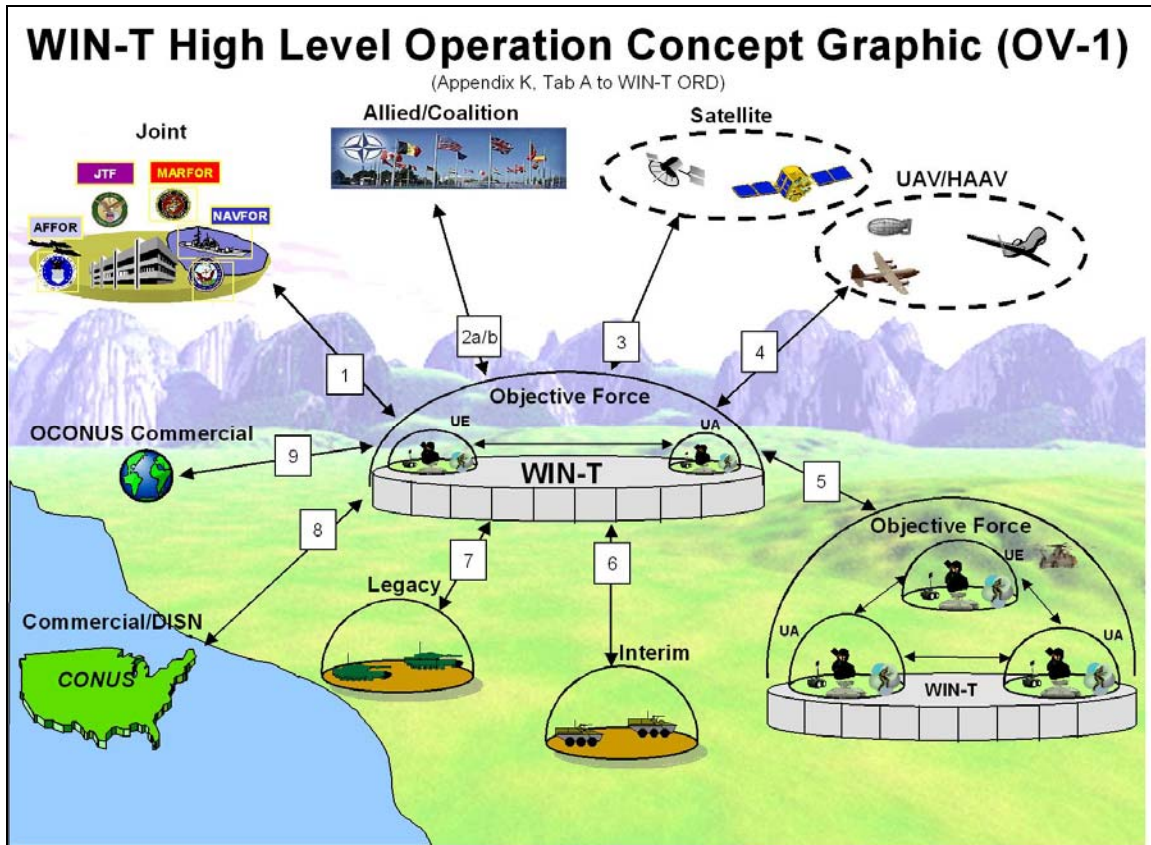


Figure 8. The GIG architecture ensures users have access to any and all resources facilitating the transition from a “need-to know” methodology to a “need-to-share” methodology. (Network-Centric Operations and Warfare Reference Model, Draft Version 0.9)

Applying the net-centric concepts to WIN-T, a GIG transport, the COI concept translates well to the Objective Force Unit of Action and Unit of Employment constructs, but only somewhat to the Allied/Coalition interchanges. Ensuring appropriate access restrictions are maintained requires proper scrutiny of personnel being registered with participating COIs. Once registered, however, the design of COIs facilitates information interchange among allies and coalition forces of all participating COIs. The traditional guards that are required to affect release of sensitive information elements are translated to the boundaries of individual COIs, which should reduce latency between United States and allied warfighters performing similar operational activities. Figure 9 is a graphical representation of WIN-T as a GIG transport detailed in the WIN-T Operational Requirements Document (ORD). (Van Dine, 27 October 2003)



research being conducted to solve the problems associated with complete implementation of FCS identity management. (Van Dine, 27 October 2003)

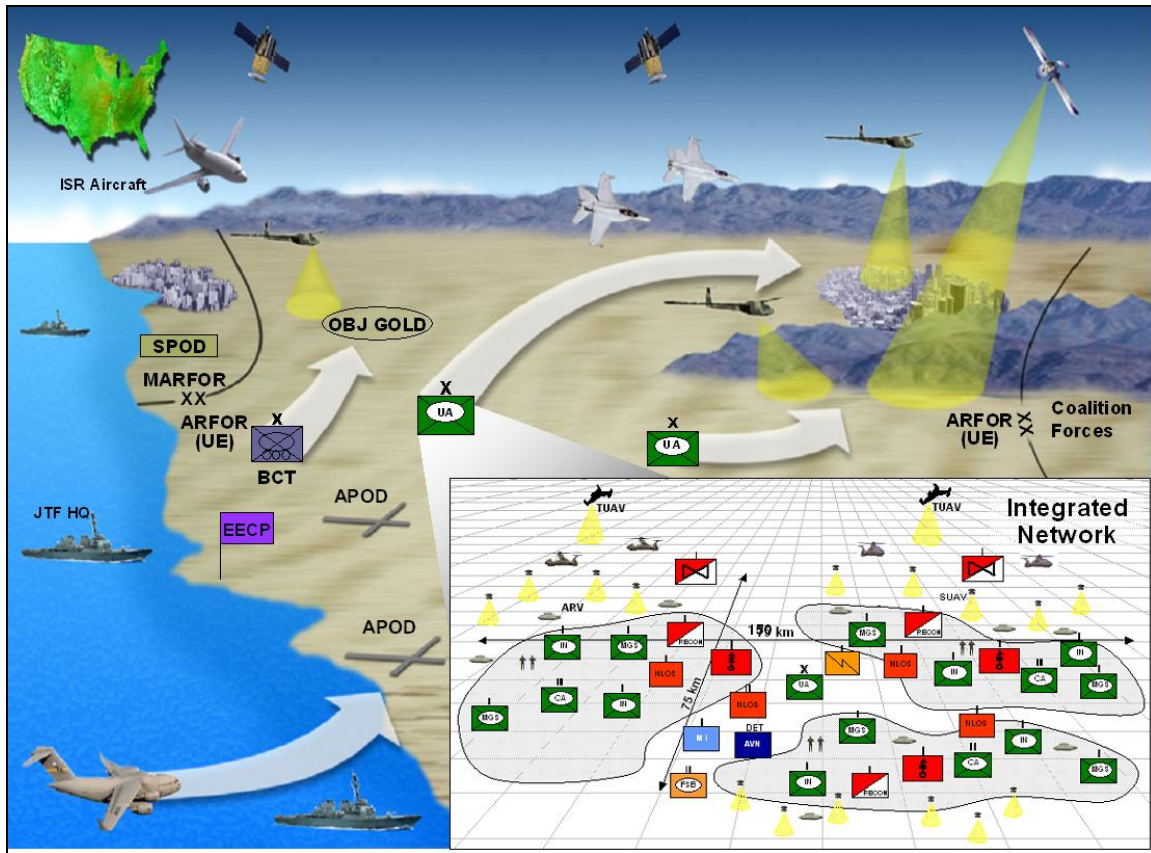


Figure 10. The FCS is a Joint networked system of systems, composed of eighteen subsystems, that uses the WIN-T platform as a backbone for its network infrastructure. (From presentation titled: “GIG from a Warfighter Perspective” authored by Wayne A. Van Dine, Jr., see Van Dine in the bibliography for more details)

F. SUMMARY

The GIG is the realization of Network-Centric Warfare applied to the concept of Grid computing. The GIG provides an extensible, service-oriented architecture that ensures maximum resource access to authorized users. Many services and applications are currently being revised to accommodate the interoperability requirements for deployment on the GIG architecture. The POC developed for this thesis demonstrates the possibility of deploying legacy systems on the GIG in their current state.

IV. DEVELOPMENT OF THE PROOF OF CONCEPT (POC) EXEMPLAR

A. INTRODUCTION

This chapter expounds on the details of the research and development of the POC, which utilizes a Java client/server application to mimic a proprietary system. The same application is then used as the basis for the Web service, which generates the data for Web-based reports and keeps the portal current without the need for manual updates. The report portal consists of a simple user interface that provides access to reports from the G1⁷.

B. RESEARCH AND DEVELOPMENT METHODOLOGY

The methodology used in this thesis research consists of the following steps:

1. Research and collect data related to standards based Web services
2. Research and collect data related to standards based UDDI registry
3. Research requirements for an alternative platform for development of Web service and UDDI registry
4. Install alternative development platform consisting of the following:
 - Linux for the operating system (OS)
 - Apache for the HTTP server using PHP as the scripting language
 - Tomcat as the Java servlet container
 - MySQL or PostgreSQL for the relational database if needed
 - Java SDKs as necessary
5. Determine/define data set to prototype for exposure by a service
6. Develop Java program that acts as the provider agent for the service
7. Implement requester agent as a PHP based Web site
8. Develop Web-based display capability for results of service request

⁷ The S1/G1/J1 are command staff sections charged with the responsibility of oversight of all personnel and administration issues.

9. Develop Web service, which includes but is not limited to the following:

- PHP configuration
- XSLT development
- SOAP implementation
- Tomcat configuration
- Apache configuration

These topics are discussed in detail in the following paragraphs and form the basis for the POC.

C. DEVELOPMENT AND TESTING PLATFORMS

As identified in chapter I, one of the goals of this research is to explore how OS independent Web services are. This is an important characteristic of Web services, identifying how flexible and extensible such an architecture is. The POC was developed on a Windows XP Pro platform with mostly cross-platform compatible technologies. Most of the development was completed on a Dell Precision M60 mobile workstation running Windows XP Pro. The following is a list of the applications and programming language technologies most relevant to the development and implementation process:

- Apache 2 HTTP Server used as testing server during development
- PHP 4.3.4 scripting language used as embedded scripting language for dynamic content within the Web site.
- PHP Extension and Application Repository (PEAR), which is installed automatically for PHP 4.3.x or greater. This is a framework to maintain and distribute open-source PHP components, which have interesting capabilities but are not Web standard.
- PEAR::SOAP, client/server SOAP implementation for PHP.
- MySQL 4.3.1 Beta-NT-Max relational database, which is used for login authentication to the Web site on the testing server.
- Sun Java J2SE 1.4.2, which was used to develop the client and server applications.
- Apache Tomcat 5.0.27 Server was used as the application server to expose the Java server application as a Web service.
- Apache SOAP 2.3.1, Apache's implementation of the SOAP 1.1 standard, was used in conjunction with Tomcat to provide the Web services.

Eclipse 3.0 integrated development framework was used to code the Java client and server applications and other required Web service classes.

Dreamweaver MX was the application used to develop the entire Web site. Although this is neither open source nor free, nor is it natively cross-platform compatible, it was chosen due to the author's familiarity. Furthermore, it provided a quick and intuitive way to create a PHP based Web site with a MySQL database backend. However, it must be noted that the default PHP/MySQL extension that is provided with Dreamweaver MX was not sufficient. Therefore, an open-source extension, PHAkt 2 MX , was used instead.

The alternative OS is Suse 9.0 Pro Linux. Although no development was conducted on Linux, testing of the applications was conducted as progress proceeded to ensure cross-platform compatibility and to test the flexibility of deployment. The alternative OS platform differed only slightly from the primary OS platform. All components necessary to employ the proof of concept on the alternate OS platform were installed. The only variations consisted of some slight differences in version. Figure 11 is a depiction of the POC deployment architecture.

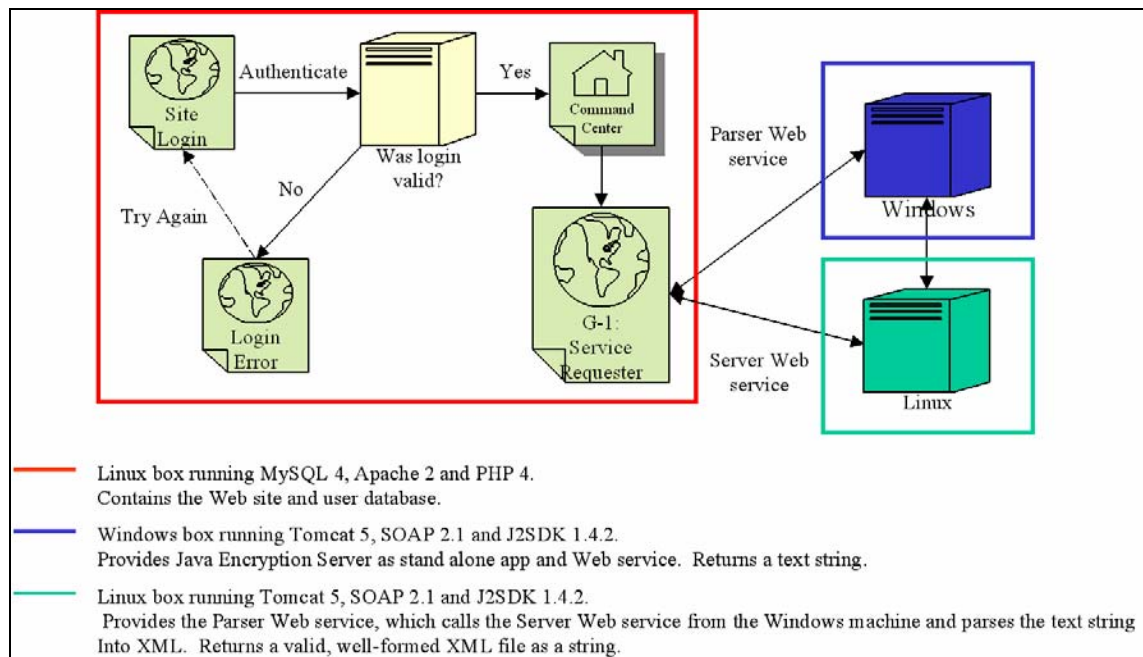


Figure 11. The target architecture for the proof of concept (POC) example depicted here is intended to demonstrate the flexibility and platform independent nature of Web services.

D. DATA REQUIREMENTS

Since the G1 portal is the focus of this POC only one data source is necessary. In the case of the POC the information accessed by the G1 portal consists of textual personnel data containing first names, last names, gender and last known duty status.

1. Personnel Reporting (G1)

In a real operational command center, personnel reporting is typically the responsibility of the S1/G1/J1 staff section. Considering the point of this research is to demonstrate how Web services can provide the commander with an improved architecture for utilizing his or her information resources, the basis for the POC involves a mock operational reporting portal representative of the type of briefing mechanism that may be used in an operational command center. In this scenario the G1 is responsible for maintaining data pertaining to the duty status of the personnel under its command. The data specifically consists of a text file that contains the first name, last name, gender and last known duty status of 15000 individuals. A notional list was generated by randomly merging lists of last names, male first names and female first names obtained from the United States Census Bureau's Web site. Once merged, two additional fields were added, gender and duty status. Construction of the final product was facilitated by importing the desired data into a Microsoft Excel spreadsheet from three separate text files called dist.all.last, dist.female.first and dist.male.first. Once in the spreadsheet the gender and duty status fields were added and filled with relevant data. After the data structure was complete the data was exported into a comma separated value (csv) text file called "PersonnelStatusFile.csv" for use with the Java client/server application.

This data is artificial and does not represent realistic reporting requirements, but rather provides a simple yet significant data source for use in the demonstration of Web service capabilities.

E. WEB SERVICE IMPLEMENTATION

1. Service Providers

The basis for the service provider is a server that was developed for this proof of concept, which stores Base-64 encoded data. The intent all along was to explore the concept of how a Web service is used to provide a programmatic interface to a proprietary data system that allows for remote accessibility by HTTP request and yields

real-time information. To demonstrate the application of such a concept a client and a server, both written in Java, were developed to represent such a proprietary system.

a. Client Application

The client is designed with three basic functions in mind: open a file on the local file system, encode and send a file to the server and retrieve a decoded file from the server. Figure 12 and Figure 13 are screenshots of the user interface that allows the user to choose a file from the local file system for viewing prior to encoding and transmitting.

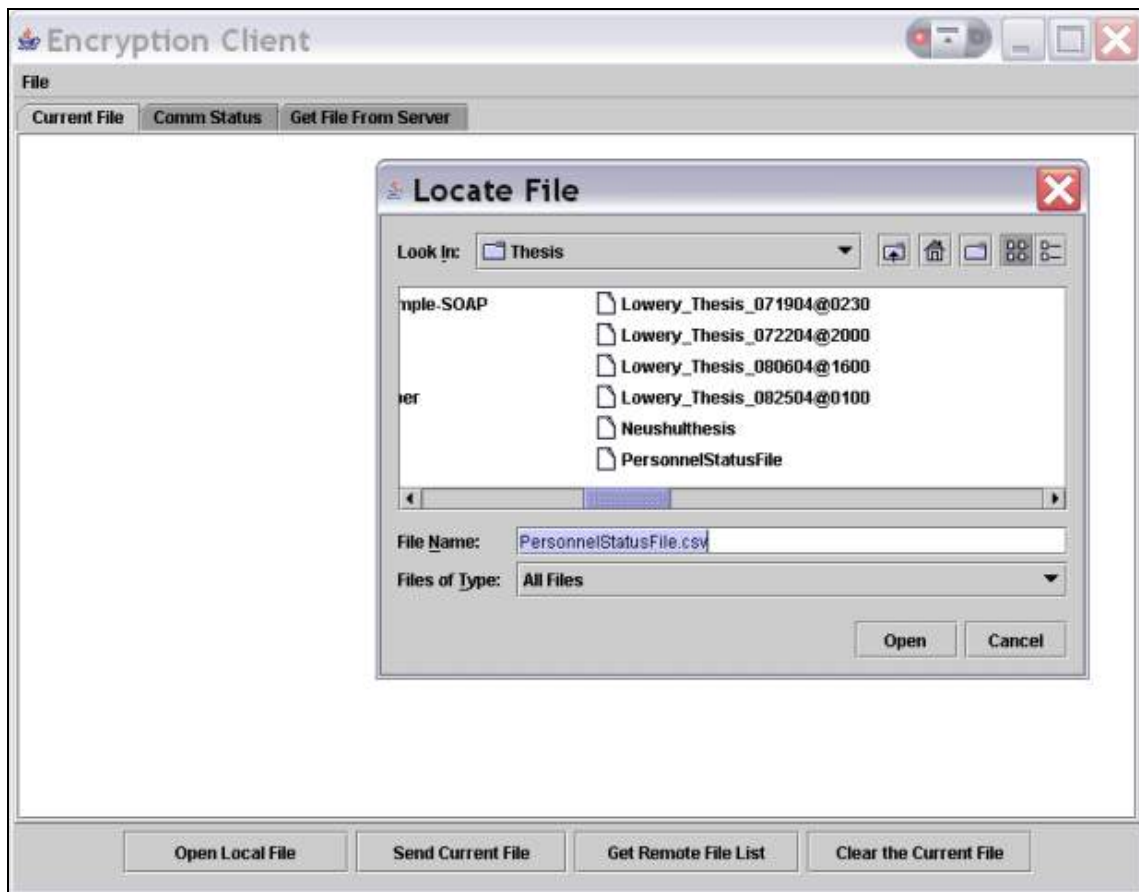


Figure 12. The "Locate File" dialog box is used to open a local file.

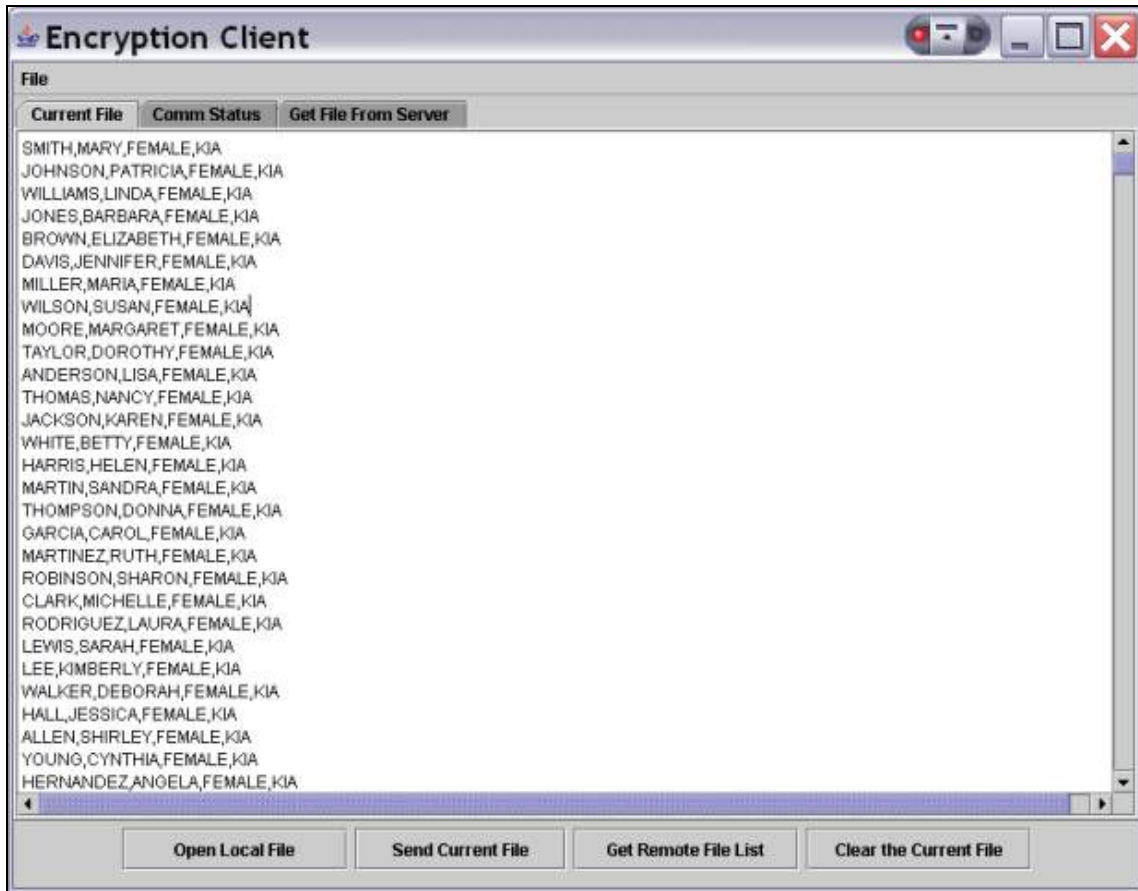


Figure 13. The “Current File” tabbed pane allows the user to view the selected comma separated file.

Once the file is opened, the user reviews it and then sends it to the server by selecting the “Send Current File” button. Selecting this button opens the connection dialog shown in Figure 14 and changes the client view to the “Comm Status” tab, which allows for monitoring of communication with the server.

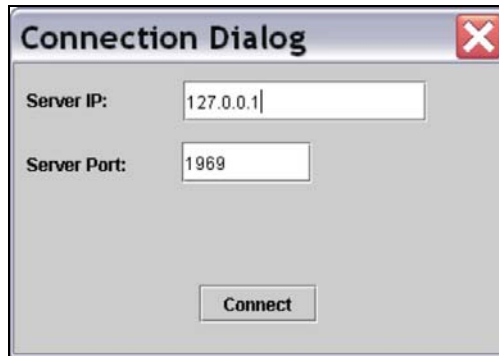


Figure 14. The connection dialog is designed to allow the user to enter the Internet Protocol address and port number of the server.

Once the server IP and port number are entered the user clicks on the “Connect” button to encode the file and send it to the server for storage. Server IP 127.0.0.1 indicates that the server is running on localhost.

To retrieve a decoded file from the server the user initially clicks on “Get Remote File List”. After processing the connection the server returns the list, which is displayed in the tabbed pane “Get File From Server.” See Figure 15 for a screenshot of the user interface. The user then enters the filename into the text box as shown in Figure 15 and clicks on the “Get this file:” button. Once the connection to the server has been processed the decoded text is displayed in the same tabbed pane. See Figure 16 for an example.

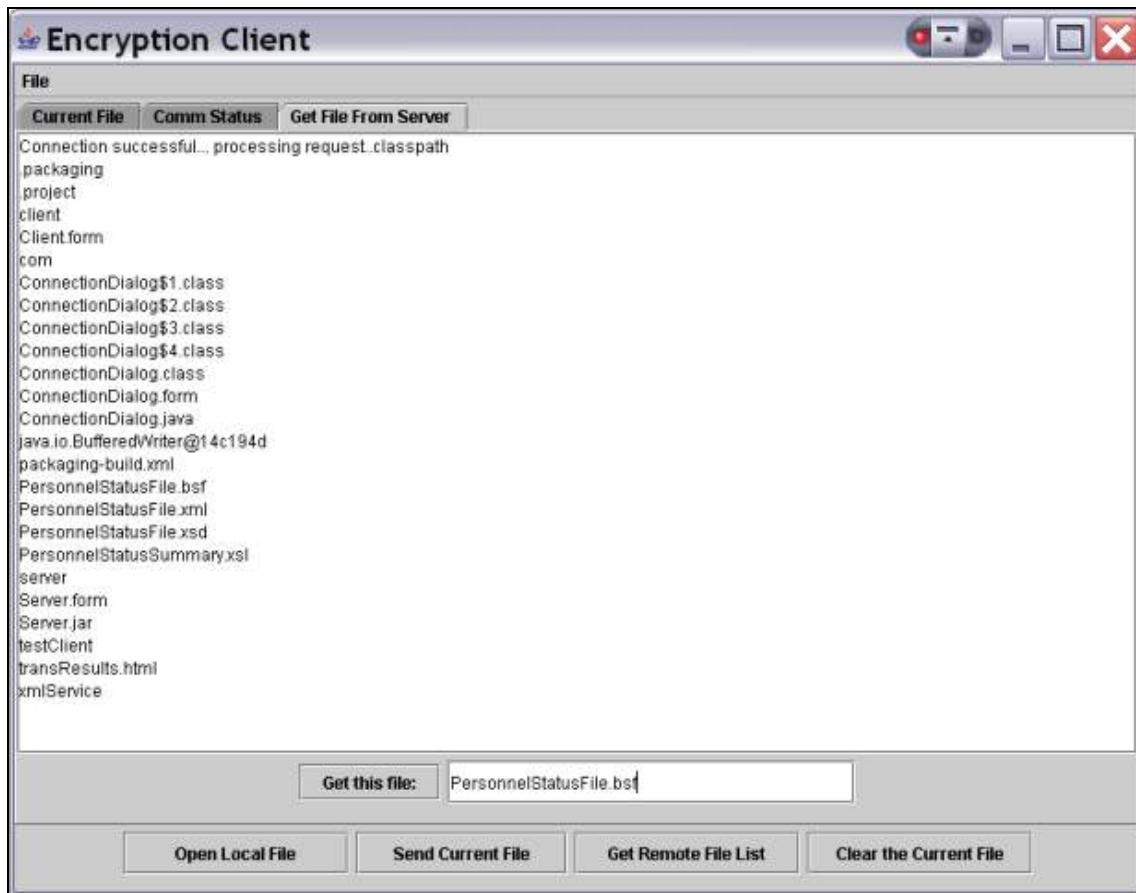


Figure 15. The “Get File From Server” tabbed pane allows the user to select a file from the server.

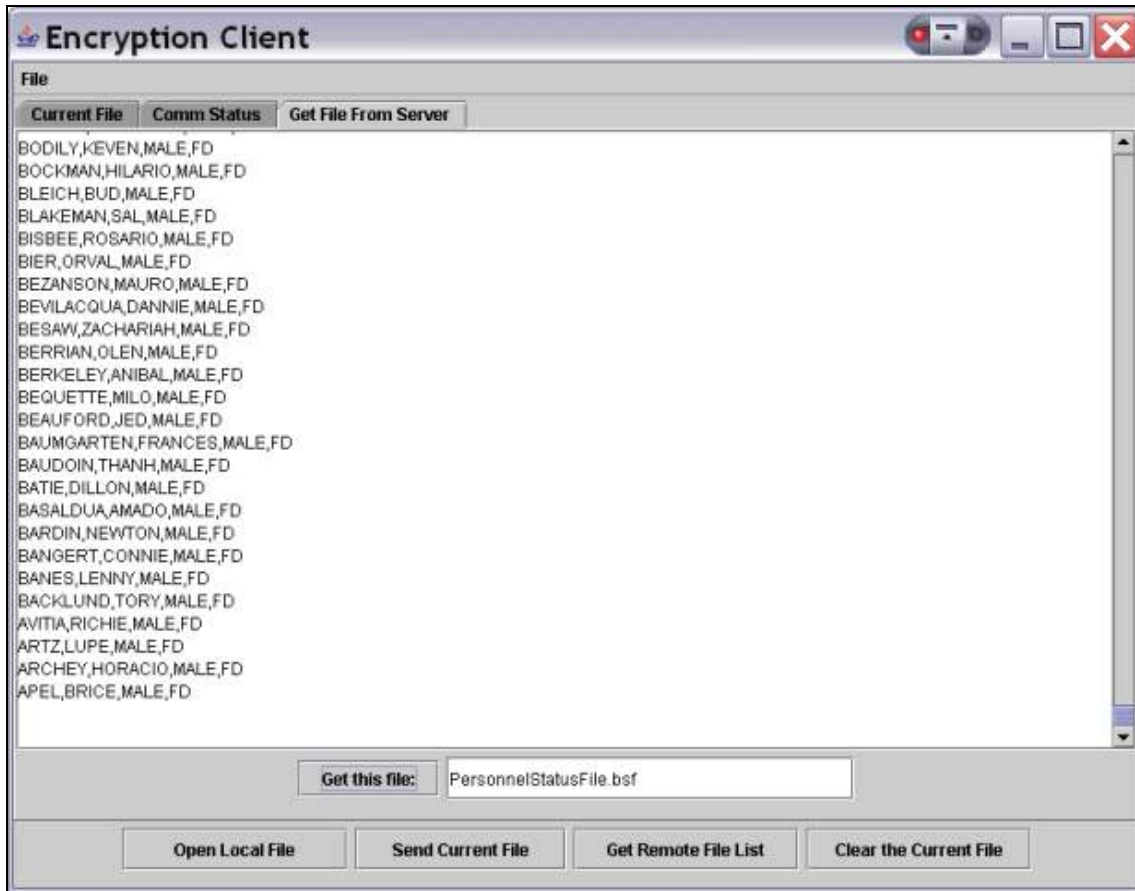


Figure 16. Retrieved data is viewed in the “Get File From Server” tabbed pane.

This process demonstrates how a proprietary client/server system is implemented to process data locally and store the data remotely. If a user needs access to the data from a remote location, the client software must be installed on their local computer in order to access the file in such a way as to retrieve the decoded data. This is conceptually similar to the UD/MIPS system discussed in chapter I.

b. Server Application

The server application is fairly simple. The original version has a graphical front-end, which can be seen in Figure 17. However, a second version was developed without a graphical front-end to allow for remote deployment in situations where a graphical context, such as an X Windows session, is not available.

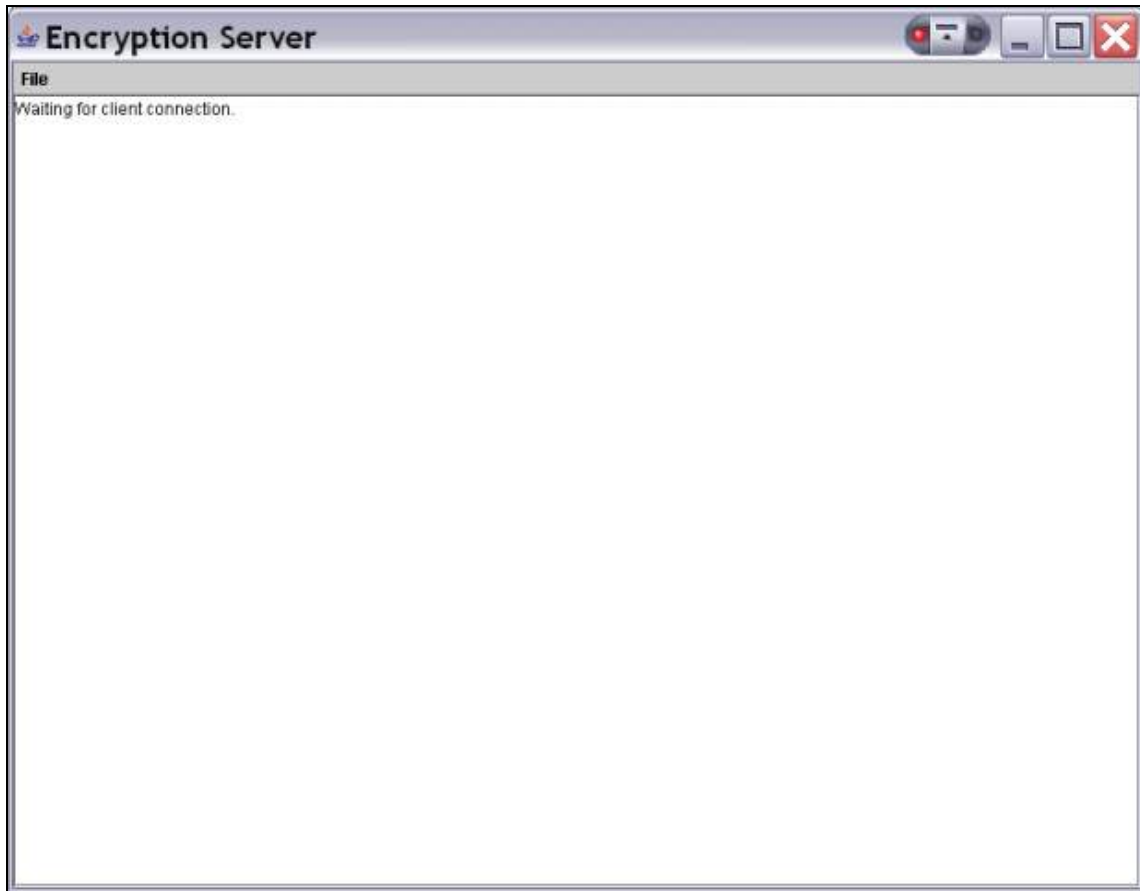


Figure 17. The server graphical front-end is displayed when the server is running.

Once running, the server waits for a connection from a client. Upon getting a connection from a client the server can take one of three actions: store an encoded file, send the client the file list or decode and transmit the requested file back to the client. The server determines which action to take based on the instructions from the client. Figure 18 shows the control statement used to facilitate this action.


```
if (clientMsg.startsWith ("gettingFileList")) {  
    sendFileListToClient ();  
} else if (clientMsg.startsWith ("gettingFile")) {  
    sendFileToClient ();  
} else if (clientMsg.startsWith ("sendingFile")) {  
    fileName = clientMsg.substring (11);  
    processEncodedFile ();  
}
```

Figure 18. The “if” statement used by the server determines which operation to perform based on the message received from the client.

The “if” statement in Figure 18 tests whether the string called clientMsg starts with one of three possibilities: “gettingFileList”, “gettingFile” or “sendingFile”. Each of these indicates the client’s intentions to the server. If the first condition is true then the server returns the file list relative to the server’s root directory. If the second condition is true then the server decodes the requested file and transmits it to the client. If the third condition is true then the server strips the first 11 characters from the clientMsg string to produce the filename. This is the name that the server uses to create a file object to save the file to the hard drive. Each condition is skipped until one is true. Figure 19 shows the file being received for storage by the server.

It is important to note that the design of this POC example is not ideal. For example, the filename is generated from the first eleven characters of the clientMsg string but in reality not all filenames are eleven characters long. The POC was developed to function properly with a specific file, with specific data in a specific format. This was done intentionally to simplify development and does not detract from the overall concept being researched.

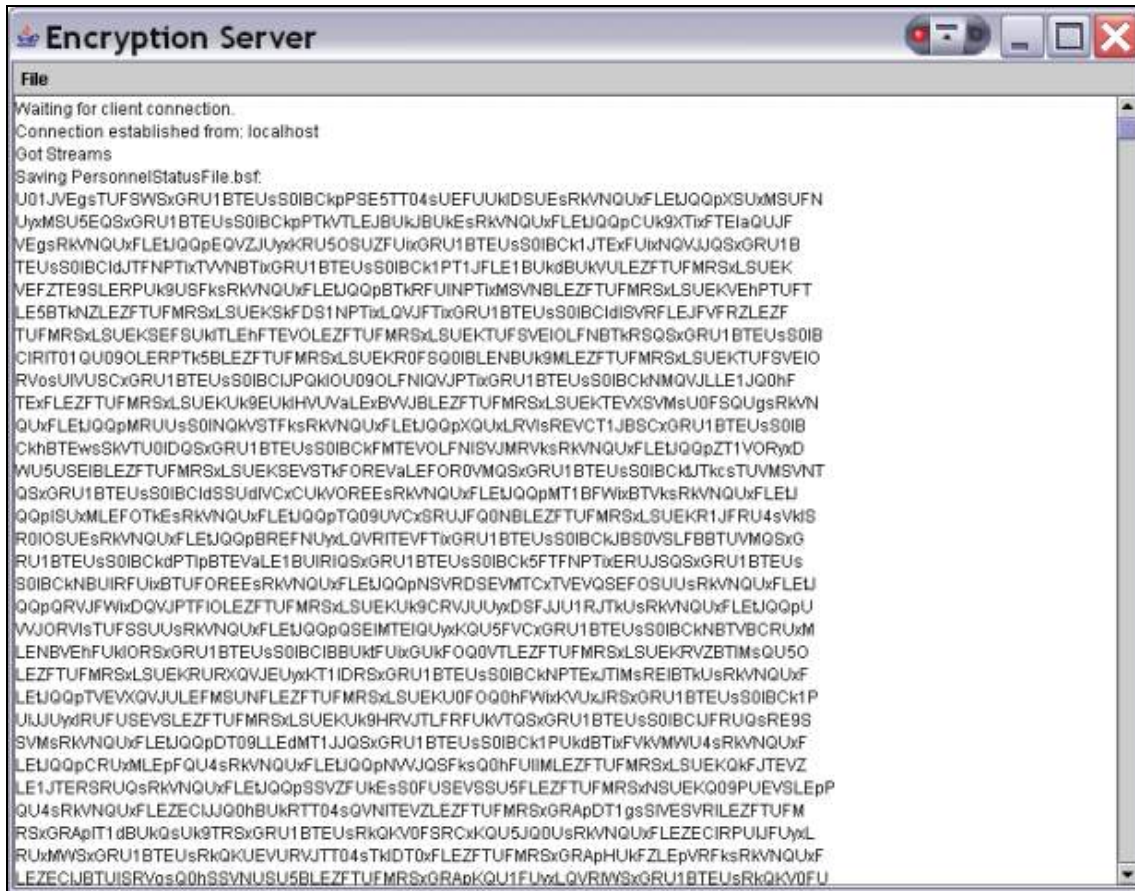


Figure 19. The server displays the Base-64 encoded file sent by the client.

When the client requests a decoded file, the server uses the method “decode”⁸ to decode the file and return clear text for transmission to the client; see Figure 20. This is important because it is the method that is exposed as a Web service and is used to return data to any Web service client, which is discussed in more detail later.

⁸ The Base-64 encoding and decoding is facilitated by using classes found in the non-standard Java package “sun.mic.”

```

public static String decode (String fileName2) {
    try {
        BufferedReader in  = new BufferedReader(
        new FileReader(fileName2));
        StringBuffer  buffer = new StringBuffer();
        String      text  = new String();

        while ((text = in.readLine ()) != null) {
            buffer.append (text + "\n");
        }
        in.close ();

        base64EncodedText  = buffer.toString ();
        clearText = new String(
        new BASE64Decoder().decodeBuffer
        base64EncodedText));
    } catch (IOException e1) {
        e1.printStackTrace ();
    }
    return clearText;
}

```

Figure 20. The server “decode” method is exposed as a Web service and provides the basic services for retrieving data from the server.

c. Server Web Service

The design of the first and most basic Web service is centered on the “decode” method of the server application. The process for deploying the server as a Web service consists of several steps. First, a package was created called “server”, which contains the server class and source files. This package was then used to create a “jar” file called Server.jar. The jar file was then placed in the \shared\lib directory relative to the Tomcat root directory. When Tomcat is restarted it is then able to provide Server.jar as an accessible service. The final step is to register the service under SOAP. SOAP is used to process requests from clients via HTTP.

Registering the service under SOAP takes several steps as well. SOAP uses a Web-based form to register services. From the SOAP Web page located in the Tomcat Web Application Manager click on the “Run” link; see Figure 21. This links to the SOAP administrator page.



Figure 21. This screenshot shows Apache-SOAP running as a Web Application under Tomcat.

Once at the administrator page click the red “Deploy” button in the menu at the left; see Figure 22. This presents a form, which when properly completed creates a service deployment descriptor file for Apache-SOAP to use for processing service requests.



Figure 22. The Apache SOAP Admin page is used to manage service deployment description configurations.

There are several options available for creating the service deployment descriptor file but the service implementation used by the POC is simple and only

requires setting four parameters. The first parameter to establish is the service ID. For the server service the ID is “urn:EncServer”; see Figure 23. Next it is necessary to list all methods that are being exposed by the service. In this case there is only one method, “decode”. This is entered as shown in Figure 23. After the methods have been listed, there is one last parameter that has to be identified, the Java provider. This is the name of the jar file that provides the service. The Java provider is server.Server and is entered as shown in Figure 24.

The screenshot shows the Apache SOAP Admin Tool interface in a Mozilla Firefox browser window. The address bar shows the URL `http://localhost:8090/soap/admin/`. The page title is "Apache SOAP Admin". On the left side, there is a green sidebar with three buttons: "List", "Deploy", and "Un-deploy". The main content area is titled "Deploy a Service". Below this title is a table with the heading "Service Deployment Descriptor Ter". The table has two columns: "Property" and "Details".

Property	Details
ID	urn:EncServer
Scope	Request
Methods	decode (Whitespace separated list of method names)
Provider Type	Java

Figure 23. This screenshot shows the service deployment descriptor ID and methods list configuration form. These parameters are used to create the service deployment descriptor file used by SOAP.

The screenshot shows the "Java Provider" configuration form. It has a green sidebar on the left. The main content area has a table with the heading "Java Provider". The table has two columns: "Provider Class" and "Static?".

Provider Class	Static?
Server.Server	No

Figure 24. The service deployment descriptor configuration form requires a Java Provider as well.

The last step is to deploy the service by clicking the “deploy” button at the bottom of the form. To view the list of services click on the red “List” button in the menu at the left. This provides a list of hyperlinks that allow viewing of the service deployment descriptors.

After completing the installation of the services and configuring the SOAP service deployment descriptor files for each service, the services can be accessed via HTTP. After being exposed as a service and implementing SOAP the server is accessible by any type of client that uses the SOAP protocol. When a client calls the server Web service it decodes the requested file and returns a string of text. It is important to make the distinction that the service provided by Server.jar is a completely separate instance of the original server application and has no affect on the original Java program. This demonstrates that data maintained by the server can be successfully accessed and retrieved in real-time by a means other than the original client application and without interfering with the server’s execution. The data is retrieved programmatically and can be further processed to meet specific needs, which is demonstrated with two other service implementations detailed later in this text. (“Apache SOAP v2.3.2 Documentation”, September 2004)

d. Parser Web Service

The parser Web service is arguably the most critical component of the architecture being implemented. In order to overcome the challenge of data format dependency, it is necessary to transform the original text into XML. Once this has occurred, the data can be further transformed with much greater flexibility using XSLT.

The parser Web service is a special interface to the original server Web service that allows a client to request a decoded file and receive XML data instead of plain text. Upon receiving a call from a client, the parser service makes a call to the server and passes the arguments as strings, which are the filename of the file to be parsed and the Uniform Resource Locator (URL) of the service. The server returns a string of text, which the parser tokenizes and parses into XML. This XML is then returned to the client. A testament to the flexibility of this architecture is that these services can be on different networks and on different OSs without any concern for interoperability.

For the purposes of the POC the parser is fairly crude and designed to work specifically with the “PersonnelStatusFile.bsrf” file. Ideally the parser is designed to be flexible enough to parse any file of the data type in question. Although this is not necessarily a trivial endeavor, it is a key stepping stone towards data format independence and interoperability. Another considerable benefit to this type of architecture is the control provided over the data. Since the data is now accessible by a means other than the original client application other service capabilities can be derived and implemented in order to automate such tasks as service discovery and data presentation.

e. Transformation Web Service

The POC demonstrates that with the basic server service clients are able to retrieve textual data and with the parser service clients are able to retrieve XML data. Finally, to demonstrate the desired end state, which is the ability to present the data in a useful manner, as information with context in real-time, a third service is implemented. The transformation Web service is another special interface to the server Web service. It is possible to have the transformation service interface with the parser service but a design decision was made to avoid chaining more than two services together. Essentially the transformation service is an extended version of the parser service. It contains the same parser capability but also contains code based on javax.xml.transform classes that facilitates the transformation process. The transformation service calls the server service in the same manner as the parser service but once it has finished parsing the returned data into XML it applies a transformation, which is determined by the XSLT. In this case the file is transformed into HTML to be displayed by a Web browser as a table with hyperlinks.

2. Service Requesters

The service requesters, or consumers, are composed of three clients that are referred to as the basic client, the parser client and the transformation client. Since Web services can be called by any SOAP supported client it is important to identify that the POC implemented the parser and transformation clients’ capabilities with two different technologies, Java and PHP. The Java implementation of these clients provides a useful tool for service testing and debugging. Once installed under the SOAP server in Tomcat,

This basic client retrieves unformatted text only, which is not very useful by itself, but represents the first stage of the complete architecture.

b. Parser Client

The parser client is essentially the same as the basic client with two changes. First the parameter array contains an additional parameter for a total of two. The additional parameter is the URL of the server Web service's SOAP RPC router. The second change is contained in the parameter list for the "call" function. The method call is changed to "getDecodedFile" and the service identifier is changed to the service being called, in this case "urn:XMLPersonnelStatusService". These are identified in the service deployment descriptor file and are used by the SOAP RPC router to properly process the service request. Everything else remains the same.

Figure 26 shows sample output of the XML data in a Mozilla based open-source Web browser called Firefox. Figure 27 shows the same output in Internet Explorer in a hierarchical form. The difference in the XML presentation is a result of browser implementation.

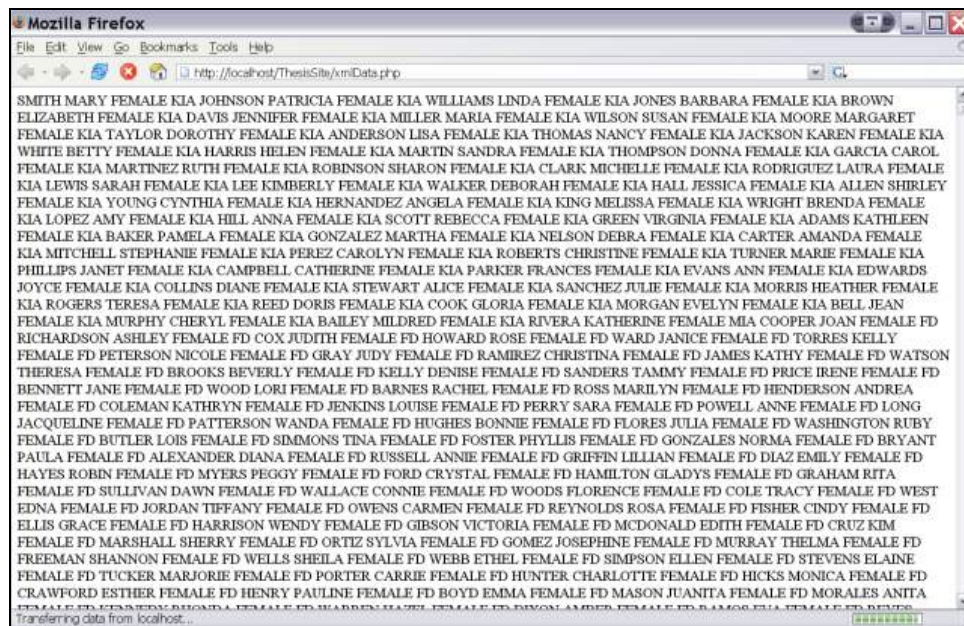


Figure 26. The second service implemented parses the text and converts it into XML. The Firefox Web browser renders resulting XML output from the parser service without the XML tree. Firefox implements XML parsing and rendering in a different manner than Internet Explorer.

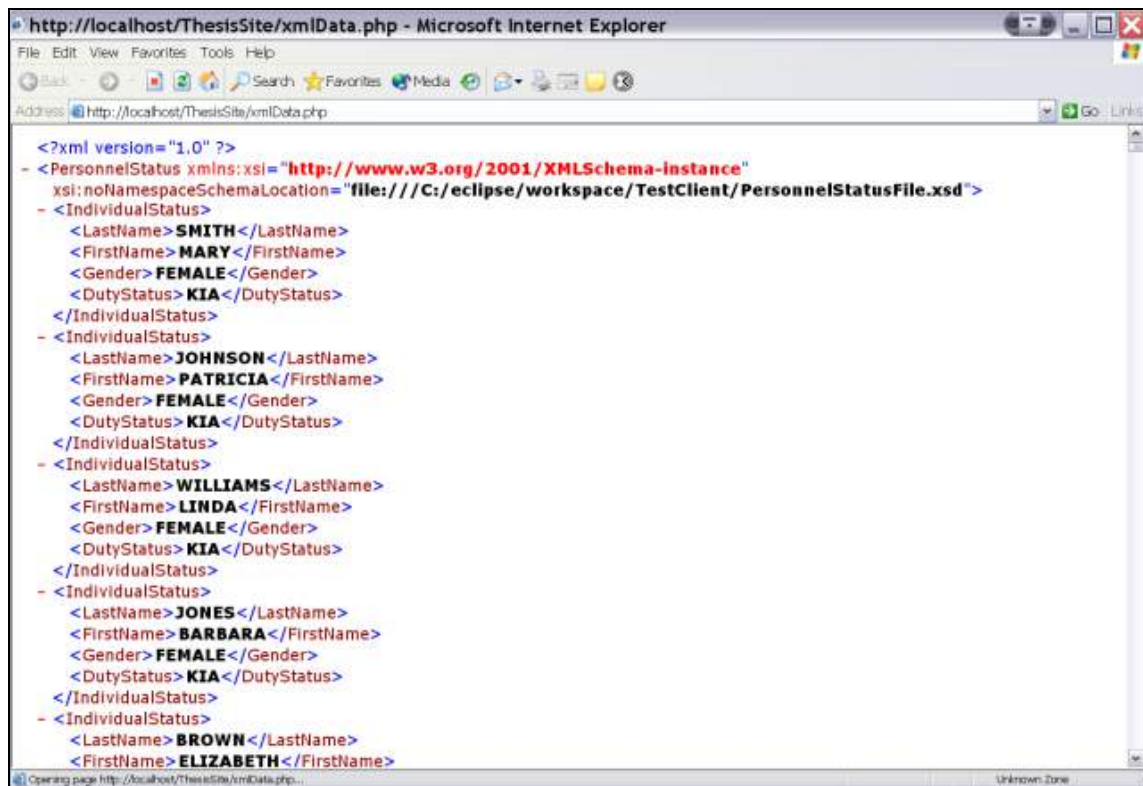


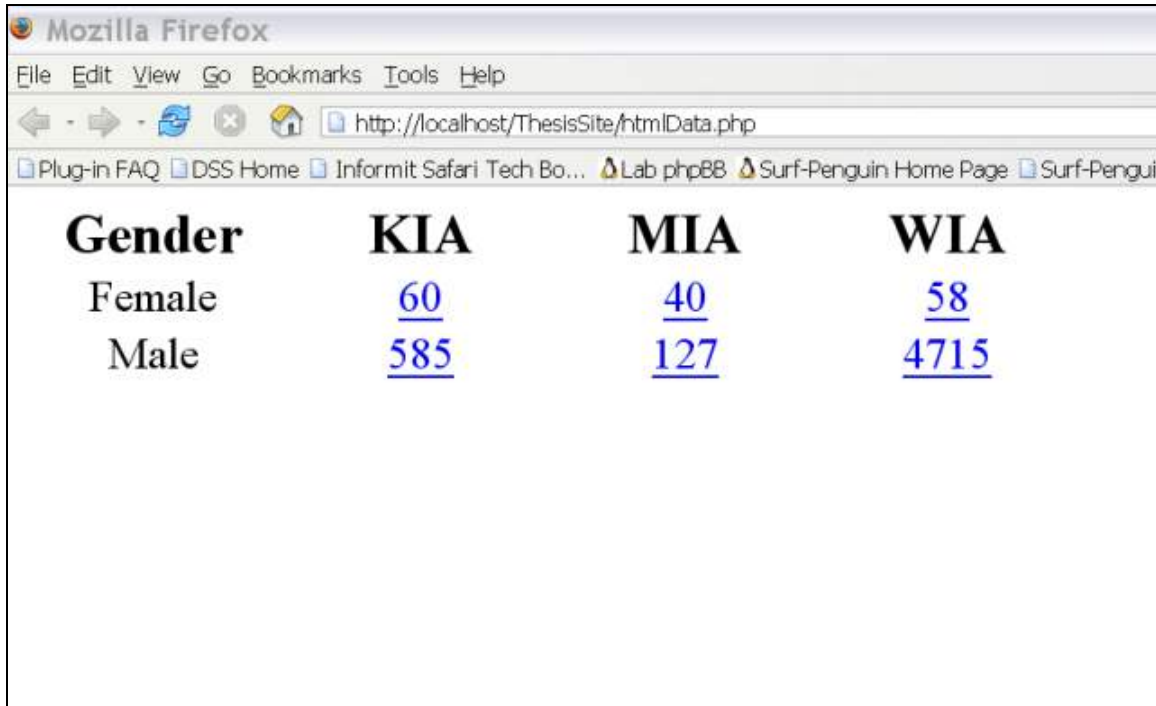
Figure 27. Although the results from the parser are the same as before, Internet Explorer renders the resulting XML output as an XML tree. This is specific to the Internet Explorer.

A simple XML schema is used to validate the data. The parser is told where the schema is located and includes the schema's Uniform Resource Identifier (URI) in the returned XML. When a validating Web browser is used, the received data is validated against the schema. This type of functionality is very important. XML Schemas can be used to impose very specific requirements on the XML data being validated. In the case of operational information this is a significant and necessary capability.

c. *Transformation Client*

The last client implementation is the transformation client. The same parameters are changed for this client as for the parser client. The only differences are the name of the method being called and the service identifier. The method name is changed to "getTransformedFile" and the service identifier is changed to

“urn:PersonnelStatusSummary”. Since the transformation Web service transforms the XML into HTML the browser renders the results as a report in an HTML table; see Figure 28.



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost/ThesisSite/htmlData.php`. The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, and Help. The address bar also shows several bookmarks: Plug-in FAQ, DSS Home, Informit Safari Tech Bo..., Lab phpBB, Surf-Penguin Home Page, and Surf-Penguin. The main content area displays an HTML table with the following data:

Gender	KIA	MIA	WIA
Female	60	40	58
Male	585	127	4715

Figure 28. The last service implemented uses XSLT to transform XML into an HTML table containing a Summary Personnel Status Report as shown in Firefox.

The transformation analyzes 15000 entities on the server side first tallying the totals based on a “Gender” XML tag and a “DutyStatus” XML tag, then returning the results in HTML. All of this information is generated “on the fly” as a request is processed. If the data changes on the server, the report generated in Figure 27 is updated automatically upon the next request.

Although not implemented in this POC each of the hyperlinks is capable of being linked to another service. For example, if the user clicks on the hyperlink “60”, a service can be called that returns a PDF version of a detailed list of all females killed in action (KIA) with all available fields of data.

F. WEB PORTAL/REPORT IMPLEMENTATION

The Web Portal is the POC’s basic platform and the primary utility used in its implementation. The overarching theme of this research is how to most effectively

provide real-time information to the commander. Web sites are very common tools for providing staffs access to remote information but that remote information typically becomes stale very quickly. Dynamic Web sites using database back-ends require considerable maintenance because they do not have direct access to the data at its origin. As information changes on the proprietary systems used in the command center, updates have to be committed to the database, but the data often first requires exporting from the original system and reformatting. These changes typically take considerable effort. The goal is to provide an architecture that can use a Web browser to access any and all systems at the commander's disposal and integrate the data into a functional tool that allows for a comprehensive perspective of the battlespace.

G. SUMMARY

The POC uses a client Java program and server Java program as a notional proprietary data architecture. The server is then exposed as a Web service to demonstrate the capabilities of a Web service implementation.

V. PROOF OF CONCEPT (POC) EXEMPLAR RESULTS

A. INTRODUCTION

The following provides information based on the results of the POC. The author provides insights gained from the research and development of the POC implementation. This chapter starts by discussing the significance of Web service flexibility by discussing the results of the deployment architecture. Next, the author discusses the impact of the use of technologies with OS independent characteristics. Thirdly, the author provides information pertaining to the interoperability of the services. Finally, the chapter discusses the significance of proper parser design determined as a result of the parser service created for the POC.

B. WEB SERVICE EMPLOYMENT

To demonstrate the flexibility of a Web service architecture all application development uses platform independent technologies and is deployed on both Microsoft Windows and Linux.

A non-graphical version of the server application written in Java runs on the Linux server in order to facilitate remote employment that does not require a graphical context, such as an X Windows session. The running of the server application does not affect the implementation of the services. The services can be designed to run from an entirely different machine. Figure 29 is a representation of just one deployment scheme the POC is capable of being deployed under. Deployment can be fairly complicated as a result of the way different OSs handle class paths. Java has a number of classes designed to assist with this issue but when deploying a distributed architecture such as the POC it can become difficult to manage.

The POC was designed on a single machine with services running from “localhost”. The final results demonstrate that the services can be exposed on different machines on a network running on different OSs. To realize the true benefits of such an architecture proper planning and design is critical for the long-term.

however, that causes the client to disconnect from the server when trying to upload a file remotely. This problem does not exist when executed on the localhost, only when networked. The client running on Windows XP is able to download and view a file from the server running on Linux without any problems. This bug is being investigated.

The Web Portal uses extremely flexible technologies. Since the Department of Defense is standardized on Microsoft software, it is important the technologies used are compatible with the Windows platform. PHP, the embedded scripting language, and MySQL, the database server, are both available for Linux and Windows. Installation and configuration is easy and well documented. The Web Portal was deployed on both OSs under the Apache HTTP server. Apache is the most active and widespread HTTP server on the Web today, as shown in Figure 30.

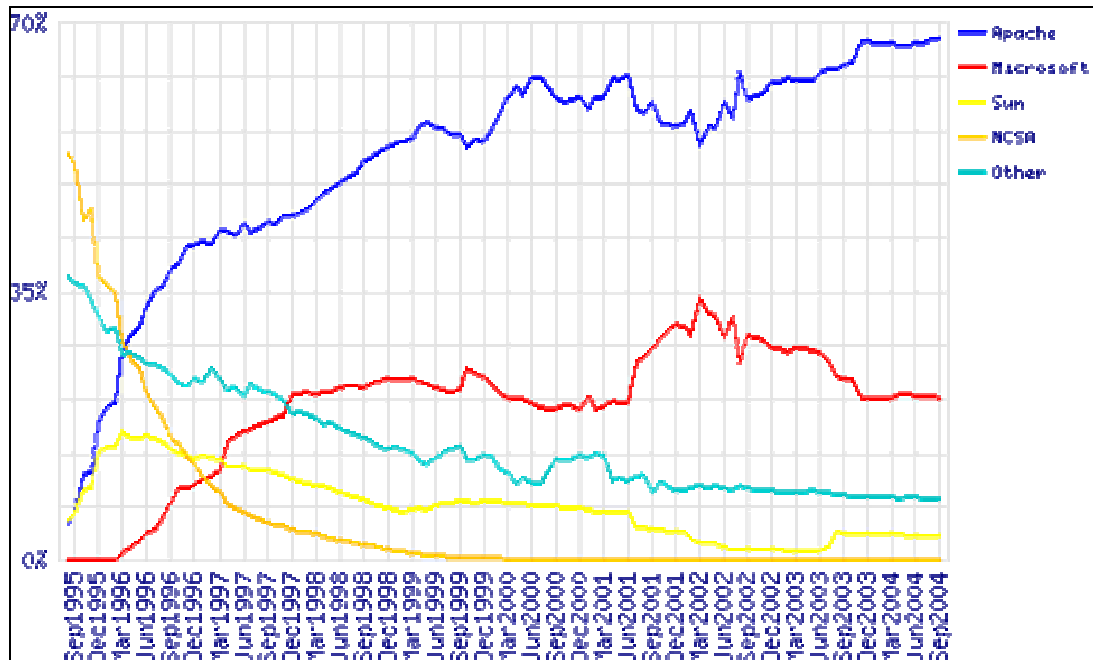


Figure 30. Much effort was made to utilize open-source technologies during the conduct of this research. The Web server of choice for the proof of concept was Apache on both Windows and Linux. These statistics represent the market share for top servers across all domains from September 1995 to September 2004, which demonstrates the ubiquity of Apache. (From: http://news.netcraft.com/archives/web_server_survey.html, September 2004).

These three technologies provide an extremely stable and robust architecture for the Web Portal. As the site was developed on the Windows machine updates were

committed to the remote Linux machine without any issues. The MySQL database consisted of a single table used to provide user access control to the Web site. The table was simply copied to the Linux version of the MySQL server and the Web site was able to use it without issue.

D. INTEROPERABILITY

Interoperability is the ability of one system to use the parts or components of another system (Merriam-Webster Online, September 2004). The POC demonstrates the validity of Web service interoperability. Although the POC was developed as a whole architecture on one machine, pieces of the architecture are deployed to multiple machines of different platforms effectively applying the concept of interoperability. Perhaps the most significant demonstration of interoperability is the use of PHP and Java to call the services. When calling the XML data service from the Web site (see Figure 31), PHP is used to call the parser service, which is written in Java, and then the parser service calls the server service and returns the results to the PHP Web page. The server service called by the parser service can also be called from within the Web site using PHP not just the parser service written in Java.



Figure 31. Each button of the G1 Web portal calls a service designed to return specific data to demonstrate the capabilities of each service. Each service represents a stepping-stone of the final product.

E. PARSER FUNCTIONALITY

In order to accomplish the desired results the POC needs the ability to parse through a string of text, identify specific characteristics associated with that string of text and break it up into separate elements or *tokenize* it. This is necessary in order to change the text into a string of XML. Each token is tagged according to criteria established by the parser. When the parser service is called, another call is made to the server service. The “PersonnelStatusFile” file, a Base-64 encoded file about 459 kilobytes in size, is read and decoded by the server service. The parser service receives a text string, tokenizes it and tags it. The resulting string is approximately 2.2 megabytes in size, which is understandable when considering the file contains 15000 individual records. This type of information is very important to consider during the design process. If the data being returned by the service is going to be significant in size, it may be useful to employ some type of compression scheme to minimize bandwidth consumption. All services must be able to be practically employed over the network.

The parser must be designed to be flexible and extensible. It should support many data processing requirements but needs to be stable and robust enough to support many requests at the same time, which is as much a function of the deployment architecture as it is design.

The parser may be employed in many different ways. In the case of the POC, the parser was implemented as a service in order to be called from the Web site and demonstrate the process of transforming textual data into a summarized, formatted report, which is the chore the transformation service performs.

F. SUMMARY

This chapter discussed the results of the implementation of the POC. Information was also provided pertaining to the impact of some of the supporting technologies.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Much is being investigated regarding interoperability specifications as the DoD transitions from current information systems to those that will be employed under the umbrella of the GIG ES architecture. In providing the services identified to meet the end user information requirements, it is crucial that future software development abides by the GIG ES interoperability specifications. When fully deployed, many of the services the GIG is intended to provide will be Web based, enabling commanders to maintain an up-to-date perspective of the battlespace. This allows for a greater focus of effort on the decision-making process and requires less effort to keep information current. Web services is one of the technologies being considered to meet these needs. A Web services architecture does not necessarily require existing applications be rewritten or redesigned in order to be deployed as Web services. Many of the services that the GIG ES is going to provide are not new services but derived from currently used systems (“The Solution”, September 2004). This research provides a proof of concept that demonstrates the possible effectiveness of leveraging a Web services architecture against currently employed legacy data systems, such as UD/MIPS, and proprietary applications. Although the POC doesn’t use an actual Marine Corps legacy system, great efforts have been taken to create a notional environment that would provide for valid analysis of the results. The POC is intended to represent a plausible mechanism a commander can employ to maintain his or her situational awareness in a modern operational command center.

The difficulties of implementation are more a function of overall architecture design than application design. If the intent is to simply take existing network architecture and an existing application and create a Web service with minimal deployment expectations, the technical challenges can be overcome with enough effort and ingenuity. Nevertheless it quickly becomes apparent that this methodology is not flexible or stable enough for the long-term if the intention is to create a service oriented architecture at an enterprise level. In other words, it has to be scalable. To ensure success in the long-term the business and technical architects must meet challenges with

detailed coordination across all encompassing organizations to coordinate standard processes for service implementation and discovery (Varhol, 30 March 2004). This ensures compatible deployment of both legacy and new applications on an infrastructure designed to support maintenance, stability and scalability.

A Web services architecture is designed to provide the necessary extensibility and flexibility to facilitate implementing existing client/server software and data systems. The basic concept is fairly easy to grasp but implementation can become very complex. In the case of the POC, implementing a Web services architecture consists of three main actions; expose a server as a service, accessible via HTTP, that can respond to SOAP requests; create a Web service deployment descriptor file used by the SOAP processor to process SOAP requests; and finally, build a client based on the Web service deployment descriptor parameters to make SOAP requests. During the development process, as the Java client and server were completed, the complexities involved in implementing the Web service quickly became apparent. The Java server application is not specifically designed for employment as a Web service but was successful nonetheless, which validates the concept. If good programming techniques are used during development of an application, implementing it as a Web service is not a problem. One such example exists in the POC. The “decode” method mentioned in chapter IV is the method that constitutes the server Web service. It is specifically designed employing a “return” statement, which facilitates the Web service implementation.

The author’s experiences deploying the server as a Web service, and creating two other services based on the results of the basic service, have led to the conclusion that although Web services technologies can be leveraged successfully against legacy and proprietary systems, long-term planning of a scalable enterprise level architecture is critical if benefits are to be realized. The plan needs to consider the impact of commercial solutions vice open-source and make every effort to adhere strictly to an open standards implementation. Open standards allow for greater interoperability and help avoid reliance on proprietary solutions.

B. RECOMMENDATIONS FOR FUTURE WORK

1. XML-Based Authentication and Authorization Technologies

The GIG is currently being developed and is ripe for research. The first suite of services, NCES, being offered is a core service set that is considered to be useful across the DoD. As progress is made, community specific services will be made available called COI services. COIs are groups of users that have common interests and goals, share mission or business processes and have agreed-upon terms of service behavior. An example of a COI is Combatant Commander Operational C2 (“The Resources”, September 2004). Many of these services have an access control requirement. There needs to be a method to facilitate necessary authentication and authorization mechanisms that are matched to the underlying architecture. Current mechanisms are difficult to maintain and service, especially on very distributed systems such as UD/MIPS. Furthermore, not all services will require the same level of control. For example, unit tables of organization (TO) are accessible as static documents via the Internet if the commodity manager provided the necessary password. That password only grants the user access to specific TOs based on their level of command. A user at command “A” cannot see the TO for command “B” and vice versa. If this capability was provided as a Web service with agreed upon tagging conventions using XML, all data could be stored as an XML database and TOs could be generated “on-the-fly” on a per-request basis. Each user logs into a Web page, which provides a list of parameters to choose from based on user identification and access rights.

The previous example is probably appropriate for sensitive but unclassified information control but other research needs to be done in the area of access control methods and encryption for classified material. The W3C currently has working groups involved in developing standards for exactly these issues. Three notable working groups are the XML Encryption Working Group, XML Key Management Working Group and the XML Signature Working Group.

The XML Encryption Working Group’s mission is to develop a means of encrypting and decrypting digital content, including complete or portions of XML documents. XML syntax is used to represent the encrypted data and provide information to enable intended recipients to decrypt it (Reagle, June 2002).

The XML Key Management Working Group's mission is to develop an XML application/protocol specification allowing clients to obtain "key" information such as values, certificates, and management or trust data, from a Web service. The basis for this specification is the XML Key Management Specification (XKMS). XKMS is composed of two components, the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS). X-KISS is used to define a *Trust* service protocol that resolves public key information contained in XML Signature elements. X-KRSS is used to define a public key registration Web service protocol that allows for public key verification across multiple Web services. (Farrell et al, September 2004)

2. Web-Service Based Common Operating Picture (COP)

Another area of research related to the realization of the GIG is a Web service based common operating picture. There are several tools available to the commander today that attempt to provide a visual perspective of friendly and enemy troops and assets, painting a picture of the battlespace. These applications typically require subject matter experts to run and interpret them with much time and resources being wasted just trying to interpret and maintain the data. The level of data integration is minimal and interoperability is a significant problem. The graphic representation is typically two dimensional and inflexible. As the GIG is fully deployed these data systems will become better integrated with previously non-interoperable systems allowing for a more robust capability. Since many of the services are going to be Web based, it only seems fitting that there is a need for future work involving a Web service COP.

Major Claude Hutton, a former student of the Naval Postgraduate School, conducted research involving the three dimensional rendering of terrain data using XML and the Virtual Reality Modeling Language (VRML)/Extensible 3D Graphics (X3D) in a Web browser. His work demonstrates the ability to query a database containing terrain data and generate a terrain model in real-time. This proof of concept is a good start towards a working Web services based common operating picture prototype. Although the GIG is determined to bring a greater level of interoperability and data integration,

without research such as this the commanders may still be left with “home grown” solutions. A Web services based COP can provide a standard platform for data integration. (Hutton, 2003)

3. Web Service Discovery

Another major issue with creating an SOA is the concern for service discovery. As the GIG becomes a reality and more and more services are deployed it is imperative that consumers are able to find what they need. Discovery is the means by which a consumer is able to locate and implement desired services and UDDI is the normative standard for implementation of such means. Certain aspects associated with service registration need careful consideration. Not all services need to be accessible to all users. Within the architecture of the GIG communities of users with similar interests called COIs are a good example of why proper UDDI implementation is so necessary. Research in this area needs to be conducted to identify how to provide access to service information based on the requestor. Going a step further with this research, there is a need to automate as much of the discovery and integration as possible. Many services, if properly designed can be discovered and integrated without much if any user interaction. This type of automation is significant from a staff perspective. If a staff has to deploy into an operational environment the services that are used in garrison will most likely still need to be available. If the services and UDDI registry are well developed and robust, proper Standing Operating Procedures (SOP) will dictate what services are virtually “plug-n-play”. When the staff establishes connectivity in the field, the basic service set established by SOP is fully functional, reducing functional downtime considerably (Bryan, 19 July 2002).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – CLIENT CLASS

```
package client;

import java.awt.*;
import java.awt.event.*;

import java.io.*;

import javax.swing.*;

/**
 * The Client class provides a graphical client that allow
 * users to encode
 * files and store them on the server, which is specifically
 * designed to work
 * with this client.
 *
 * @author David Lowery
 * @version 1.0
 */
public class Client extends javax.swing.JFrame {
    //~ Static fields/initializers -----

    private static JTextArea getFileTextArea;
    private static JTextArea serverCommTextArea;

    //~ Instance fields -----

    private JButton          getThisFileBtn;
    private JTextField       getFileTextField;
    private JScrollPane      getFileScrollPane;
    private JPanel           getFileBtnPanel;
    private JPanel           getFilePanel;
    private JButton          clearFileBtn;
    private JEditorPane      fileTextEditorPane;
    private JButton          getFileBtn;
    private JButton          sendFileBtn;
    private JScrollPane      serverCommScrollPane;
    private JScrollPane      fileScrollPane;
    private JButton          fileOpenBtn;
    private JMenuItem        getFileMenuItem;
    private JPanel           btnPanel;
    private JTabbedPane      clientTabbedPane;
    private JPanel           mainPanel;
}
```

```

private JMenuItem      exitMenuItem;
private JSeparator      jSeparator2;
private JMenuItem      sendToServerMenuItem;
private JMenuItem      openFileMenuItem;
private JMenu           fileMenu;
private JMenuBar        jMenuBar1;
private File            file;
private JFileChooser     jFileChooser1;
private StringBuffer    buffer;
private String          conString;
private BufferedReader   in;
private String          fileText;

//~ Constructors -----

/**
 * Constructor for a new Client object.
 */
public Client () {
    initGUI ();
}

//~ Methods -----

/**
 * Event handler - invoked when JButton fileOpenBtn is
 * selected. Calls
 * method openFile.
 *
 * @param evt Event to be handled.
 */
protected void fileOpenBtnActionPerformed (ActionEvent evt)
{
    openFile ();
}

/**
 * Event handler - invoked when JMenuItem
 * sendToServerMenuItem from Menu is
 * selected. Calls method sendFile.
 *
 * @param evt Event to be handled.
 */
protected void sendToServerMenuItemActionPerformed
(ActionEvent evt) {
    sendFile ();
}

```

```

/**
 * Event handler - invoked when JMenuItem exitMenuItem from
 * Menu is
 * selected. Terminates client application.
 *
 * @param evt Event to be handled.
 */
protected void exitMenuItemActionPerformed
    (ActionEvent evt) {
    System.exit (0);
}

/**
 * Method initGUI initializes the GUI.
 */
protected void initGUI () {
    try {
        mainPanel = new JPanel();
        clientTabbedPane = new JTabbedPane();
        fileScrollPane = new JScrollPane();
        fileTextEditorPane = new JEditorPane();
        serverCommScrollPane = new JScrollPane();
        serverCommTextArea = new JTextArea();
        getFilePanel = new JPanel();
        getFileBtnPanel = new JPanel();
        getThisFileBtn = new JButton();
        getFileTextField = new JTextField();
        getFileScrollPane = new JScrollPane();
        getFileTextArea = new JTextArea();
        btnPanel = new JPanel();
        fileOpenBtn = new JButton();
        sendFileBtn = new JButton();
        getFileBtn = new JButton();
        clearFileBtn = new JButton();
        jMenuBar1 = new JMenuBar();
        fileMenu = new JMenu();
        openFileMenuItem = new JMenuItem();
        sendToServerMenuItem = new JMenuItem();
        getFileMenuItem = new JMenuItem();
        jSeparator2 = new JSeparator();
        exitMenuItem = new JMenuItem();

        BorderLayout thisLayout = new BorderLayout();

```

```

this.getContentPane ().setLayout (thisLayout);
thisLayout.setHgap (0);
thisLayout.setVgap (0);
this.setDefaultCloseOperation
    (WindowConstants.EXIT_ON_CLOSE);
Client.setDefaultLookAndFeelDecorated (true);
this.setTitle ("Encryption Client");
this.setResizable (true);
this.setSize (new java.awt.Dimension(807, 627));
this.setLocation (new java.awt.Point(400, 210));

BorderLayout mainPanelLayout = new
    BorderLayout();
mainPanel.setLayout (mainPanelLayout);
mainPanelLayout.setHgap (0);
mainPanelLayout.setVgap (0);
mainPanel.setPreferredSize (new
    java.awt.Dimension(800, 600));
this.getContentPane ().add (mainPanel,
    BorderLayout.CENTER);
clientTabbedPane.setPreferredSize (
    new java.awt.Dimension(800, 600));
mainPanel.add (clientTabbedPane,
    BorderLayout.CENTER);
fileScrollPane.setHorizontalScrollBarPolicy (
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
fileScrollPane.setVerticalScrollBarPolicy (
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
fileScrollPane.setPreferredSize
    (new java.awt.Dimension(795,
        530));
clientTabbedPane.add (fileScrollPane);
clientTabbedPane.setTitleAt (0, "Current File");
fileTextEditorPane.setPreferredSize (
    new java.awt.Dimension(792, 481));
fileScrollPane.add (fileTextEditorPane);
fileScrollPane.setViewportViewView
    (fileTextEditorPane);
clientTabbedPane.add (serverCommScrollPane);
clientTabbedPane.setTitleAt (1, "Comm Status");
serverCommTextArea.setEditable (true);
serverCommTextArea.setEnabled (true);
serverCommTextArea.setToolTipText
    ("Area to monitor communication with server.");
serverCommScrollPane.add (serverCommTextArea);
serverCommScrollPane.setViewportViewView
    (serverCommTextArea);

```

```

BorderLayout getFilePanelLayout = new
                                orderLayout();
getFilePanel.setLayout (getFilePanelLayout);
getFilePanelLayout.setHgap (0);
getFilePanelLayout.setVgap (0);
clientTabbedPane.add (getFilePanel);
clientTabbedPane.setTitleAt
                    (2, "Get File From Server");
getFileBtnPanel.setPreferredSize
                    (new java.awt.Dimension(795, 47));
getFilePanel.add (getFileBtnPanel,
                    BorderLayout.SOUTH);
getThisFileBtn.setText ("Get this file:");
getThisFileBtn.setToolTipText
("Sends request to server for the chosen file.");
getThisFileBtn.setPreferredSize
                    (new java.awt.Dimension(100, 26));
getFileBtnPanel.add (getThisFileBtn);
getThisFileBtn.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {
            getThisFileBtnActionPerformed (evt);
        }
    });
getFileTextField.setToolTipText
    ("Type the name of the file you wish to
choose exactly as it appears above.");
getFileTextField.setPreferredSize
                    (new java.awt.Dimension(289, 30));
getFileBtnPanel.add (getFileTextField);
getFileTextField.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {
            getFileTextFieldActionPerformed (evt);
        }
    });
getFilePanel.add (getFileScrollPane,
                    BorderLayout.CENTER);
getFileScrollPane.add (getFileTextArea);
getFileScrollPane.setViewportViewView
                    (getFileTextArea);

FlowLayout btnPanelLayout = new FlowLayout();
btnPanel.setLayout (btnPanelLayout);

```

```

btnPanelLayout.setAlignment (FlowLayout.CENTER);
btnPanelLayout.setHgap (5);
btnPanelLayout.setVgap (5);
btnPanel.setPreferredSize (new
    java.awt.Dimension(800, 47));
btnPanel.setMaximumSize (new
    java.awt.Dimension(800, 90));
mainPanel.add (btnPanel, BorderLayout.SOUTH);
fileOpenBtn.setText ("Open Local File");
fileOpenBtn.setBorderPainted (true);
fileOpenBtn.setToolTipText
    ("Click to open a file on this machine");
fileOpenBtn.setPreferredSize (new
    java.awt.Dimension(160, 30));
btnPanel.add (fileOpenBtn);
fileOpenBtn.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {
            fileOpenBtnActionPerformed (evt);
        }
    });
sendFileBtn.setText ("Send Current File");
sendFileBtn.setToolTipText
    ("Encrypts and sends local file to server
for storage.");
sendFileBtn.setPreferredSize (new
    java.awt.Dimension(150, 30));
btnPanel.add (sendFileBtn);
sendFileBtn.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {
            sendFileBtnActionPerformed (evt);
        }
    });
getFileBtn.setText ("Get Remote File List");
getFileBtn.setToolTipText
    ("Gets and displays file that has been
decrypted by server.");
getFileBtn.setPreferredSize (new
    java.awt.Dimension(160, 30));
btnPanel.add (getFileBtn);
getFileBtn.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {

```

```

        getFileBtnActionPerformed (evt);
    }
    });
clearFileBtn.setText ("Clear the Current File");
clearFileBtn.setToolTipText ("Clears the text
                               editor view.");
clearFileBtn.setPreferredSize (new
                                java.awt.Dimension(160, 30));
btnPanel.add (clearFileBtn);
clearFileBtn.addActionListener (
    new ActionListener() {
        public void actionPerformed
            (ActionEvent evt) {
            clearFileBtnActionPerformed (evt);
        }
    });
setJMenuBar (jMenuBar1);
fileMenu.setText ("File");
fileMenu.setVisible (true);
jMenuBar1.add (fileMenu);
openFileMenuItem.setText ("Open Local File");
openFileMenuItem.setVisible (true);
openFileMenuItem.setToolTipText
    ("Opens a file on this
     machine.");
openFileMenuItem.setBounds
    (new java.awt.Rectangle(5, 5,
                             60, 30));
fileMenu.add (openFileMenuItem);
openFileMenuItem.addActionListener (
    new ActionListener() {
        public void actionPerformed
(ActionEvent evt) {
            openFileMenuItemActionPerformed
(evt);
        }
    });
sendToServerMenuItem.setText ("Send File to
Server");
sendToServerMenuItem.setVisible (true);

sendToServerMenuItem.setToolTipText (
    "Encrypts and sends local file to server for
storage.");
sendToServerMenuItem.setBounds (
    new java.awt.Rectangle(5, 5, 60, 30));
fileMenu.add (sendToServerMenuItem);

```

```

        sendToServerMenuItem.addActionListener (
            new ActionListener() {
                public void actionPerformed
                    (ActionEvent evt) {
                    sendToServerMenuItemActionPerformed (evt);
                }
            });
        getFileMenuItem.setText ("Get File from Server");
        getFileMenuItem.setToolTipText (
            "Gets and displays file that has been
            decrypted by server.");
        fileMenu.add (getFileMenuItem);
        jSeparator2.setLayout (null);
        jSeparator2.setVisible (true);
        jSeparator2.setBounds (new java.awt.Rectangle(5,
            5, 60, 30));
        fileMenu.add (jSeparator2);
        exitMenuItem.setText ("Exit");
        exitMenuItem.setVisible (true);
        exitMenuItem.setToolTipText ("Exits
            application.");
        exitMenuItem.setBounds (new java.awt.Rectangle(5,
            5, 60, 30));
        fileMenu.add (exitMenuItem);
        exitMenuItem.addActionListener (
            new ActionListener() {
                public void actionPerformed
                    (ActionEvent evt) {
                    exitMenuItemActionPerformed (evt);
                }
            });
    } catch (Exception e) {
        e.printStackTrace ();
    }
}

/**
 * Method openFile switches view to fileScrollPane and
 * creates a file
 * chooser dialog.
 */
protected void openFile () {
    clientTabbedPane.setSelectedComponent
(fileScrollPane);
    jFileChooser1 = new JFileChooser();
    jFileChooser1.setDialogTitle ("Locate File");
    jFileChooser1.setToolTipText ("Choose a file.");

```



```

        /*
        * remove or comment out the next two lines to disable
        * the default
        * selection or change the name of the default file if
        * desired.
        */
        int result = jFileChooser1.showOpenDialog (this);
        file = new File("D:\\My
Documents\\Thesis\\PersonnelStatusFile.csv");
        jFileChooser1.setCurrentDirectory (file);
        jFileChooser1.setSelectedFile (file);

        if (result == JFileChooser.CANCEL_OPTION) {
            return;
        }
        file = jFileChooser1.getSelectedFile ();

        try {
            in          = new BufferedReader(new
FileReader(file));
            buffer      = new StringBuffer();

            String text;
            fileTextEditorPane.setText ("");

            while ((text = in.readLine ()) != null)
                buffer.append (text + "\n");

            fileText = buffer.toString ();
            fileTextEditorPane.setText (fileText);
            in.close ();
        } catch (IOException e1) {
            e1.printStackTrace ();
        }
    }

    /**
    * Method main initializes the client by calling method
    * showGUI.
    *
    * @param args Default parameter for method main.
    */
    public static void main (String [] args) {
        showGUI ();
    }

```

```

    /**
     * Method showGUI creates a new instance of this class and
     * shows it inside
     * a new JFrame.
     */
    protected static void showGUI () {
        try {
            Client inst = new Client();
            inst.setVisible (true);
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }

    /**
     * Event handler - invoked when JButton sendFileBtn is
     * selected. Calls
     * method sendFile.
     *
     * @param evt Event to be handled.
     */
    protected void sendFileBtnActionPerformed (ActionEvent evt)
    {
        sendFile ();
    }

    /**
     * Method sendFile switches view to serverCommScrollPane and
    creates a
     * Connection object.
     */
    protected void sendFile () {
        clientTabbedPane.setSelectedComponent
(serverCommScrollPane);

        Connection conDiag = new Connection(fileText, file,
2);
        conDiag.setVisible (true);
    }

```

```

    /**
     * Event handler - invoked when JMenuItem openFileMenuItem
     * from Menu is
     * selected. Calls method openFile.
     *
     * @param evt Event to be handled.
     */
    protected void openFileMenuItemActionPerformed (ActionEvent
evt) {
        openFile ();
    }

    /**
     * Event handler - invoked when JButton clearFileBtn is
     * selected. Calls
     * method clear.
     *
     * @param evt Event to be handled.
     */
    protected void clearFileBtnActionPerformed (ActionEvent
evt) {
        clear ();
    }

    /**
     * Method clear resets the fileTextEditorPane to null.
     */
    protected void clear () {
        clientTabbedPane.setSelectedComponent
(fileScrollPane);
        fileTextEditorPane.setText ("");
    }

    /**
     * Method setServerCommTextArea sets the text of the
     * serverCommScrollPane to the value of the "conString2"
     * parameter. This is used to display
     * the communication status between the server and client.
     *
     * @param conString2 String to be displayed.
     */
    protected static void setServerCommTextArea (String
conString2) {
        serverCommTextArea.append (conString2);
    }

```

```

    /**
     * Event handler - invoked when JButton getFileBtn is
     * selected. Calls
     * method getFile.
     *
     * @param evt Event to be handled.
     */
    protected void getFileBtnActionPerformed (ActionEvent evt)
{
    getFile ();
}

    /**
     * Method getFile switches to the getFilePanel and creates
     * a Connection
     * object.
     */
    protected void getFile () {
        clientTabbedPane.setSelectedComponent (getFilePanel);

        Connection conDiag = new Connection(1);
        conDiag.setVisible (true);
    }

    /**
     * Method setGetFileTextArea sets the text of the
     * getFileTextArea to the value of the "files" parameter.
     * This is used to display a file list
     * returned from the server.
     *
     * @param files String to be displayed as file list.
     */
    protected static void setGetFileTextArea (String files) {
        if (files == "") {
            getFileTextArea.setText ("");
        } else {
            getFileTextArea.append (files);
        }
    }

    /**
     * Event handler - invoked when JButton getThisFileBtn is
     * selected. Calls method sendFileName.
     *
     * @param evt Event to be handled.
     */

```

```

    protected void getThisFileBtnActionPerformed (ActionEvent
evt) {
        sendFileName ();
    }

    /**
     * Event handler - invoked when JTextField getFileTextField
     * is invoked by pressing the enter key. Calls method
     * sendFileName.
     *
     * @param evt Event to be handled.
     */
    protected void getFileTextFieldActionPerformed (ActionEvent
evt) {
        sendFileName ();
    }

    /**
     * Method sendFileName creates a Connection object with a
     * filename argument.
     */
    protected void sendFileName () {
        Connection conDiag = new
Connection(getFileTextField.getText (), 3);
        conDiag.setVisible (true);
    }
}

```

APPENDIX B – CONNECTION CLASS

```
package client;

import java.awt.*;
import java.awt.event.*;

import java.io.*;

import java.net.*;

import javax.swing.*;

/**
 * The Connection class contains all the methods to manage and
 * process the connection for the client.
 *
 * @author David Lowery
 * @version 1.0
 */
public class Connection extends javax.swing.JDialog {
    //~ Instance fields -----

    private JButton      connectBtn;
    private JPanel       btnPanel;
    private JLabel       portLabel;
    private JLabel       addressLabel;
    private JTextField   portTextField;
    private JPanel       textFieldsPanel;
    private JTextField   addressTextField;
    private String       conStr;
    private Socket       conn;
    private BufferedReader bufferedInput;
    private BufferedWriter bufferedOutput;
    private String       fileText;
    private String       msg;
    private String       fileName;
    private File         file;
    private byte []      cipherText;
    private String       base64EncodedText;
    private int          switchKey;
    private String       serverMsg;
    protected String     ip;
    protected int        port;
    private String       fileToGet;
```

```

private StringBuffer    msgBuffer;

//~ Constructors -----

/**
 * Default constructor for a Connection object.
 */
public Connection () {
    makeConnection ();
}

/**
 * Constructor to build Connection object with the option
 * parameter.
 *
 * @param option Determines which request to send to the
 * client upon connection.
 */
public Connection (int option) {
    switchKey = option;
    initGUI ();
}

/**
 * Constructor to build Connection object with a String
 * parameter containing file text, a File object file
 * parameter and int option parameter.
 *
 * @param fileText String of text from the selected file.
 * @param file File object representing the selected file.
 * @param option Determines which request to send to the
 * client upon connection.
 */
public Connection (String fileText, File file, int option)
{
    this.fileText      = fileText;
    this.fileName      = file.getName ();
    this.switchKey     = option;

    initGUI ();
}

```

```

/**
 * Constructor to build a Connection object with a String
 * filename
 * parameter and an option parameter.
 *
 * @param str Filename to be retrieved.
 * @param option Determines which request to send to the
 * client upon connection.
 */
public Connection (String str, int option) {
    fileToGet      = str;
    switchKey      = option;
    initGUI ();
}

//~ Methods -----

/**
 * Method initGUI initializes the connection dialog box.
 */
protected void initGUI () {
    try {
        textFieldsPanel      = new JPanel();
        addressLabel         = new JLabel();
        addressTextField      = new JTextField();
        portLabel            = new JLabel();
        portTextField         = new JTextField();
        btnPanel             = new JPanel();
        connectBtn           = new JButton();

        BorderLayout thisLayout = new BorderLayout();
        this.getContentPane ().setLayout (thisLayout);
        thisLayout.setHgap (20);
        thisLayout.setVgap (20);
        this.setTitle ("Connection Dialog");
        this.setResizable (false);
        this.setModal (false);
        this.setName ("Connection Dialog");
        this.setSize (new java.awt.Dimension(342, 241));
        this.setLocation (new java.awt.Point(500, 400));
        this.setVisible (true);
        textFieldsPanel.setLayout (null);
        textFieldsPanel.setPreferredSize
            (new java.awt.Dimension(342,
                                     151));
        this.getContentPane ().add (textFieldsPanel,
BorderLayout.NORTH);

```



```

        addressLabel.setText ("Server IP:");
        addressLabel.setPreferredSize (new
java.awt.Dimension(60, 31));
        addressLabel.setBounds (new java.awt.Rectangle(9,
10, 60, 31));

        textFieldsPanel.add (addressLabel);

        // addressTextField.setText("127.0.0.1");
        addressTextField.setToolTipText (
            "Enter valid IP address in the format
192.168.0.0");

        addressTextField.setPreferredSize
            (new java.awt.Dimension(167,
                28));
        addressTextField.setBounds (
            new java.awt.Rectangle(116, 12, 167, 28));
        textFieldsPanel.add (addressTextField);
        addressTextField.addActionListener (
            new ActionListener() {
                public void actionPerformed
(ActionEvent evt) {
                    conActionPerformed (evt);
                }
            });
        portLabel.setText ("Server Port:");
        portLabel.setPreferredSize (new
java.awt.Dimension(77, 30));
        portLabel.setBounds (new java.awt.Rectangle(9,
54, 77, 30));

        textFieldsPanel.add (portLabel);
        portTextField.setText ("1969");
        portTextField.setToolTipText ("Enter valid port
number");

        portTextField.setPreferredSize (new
java.awt.Dimension(89, 28));
        portTextField.setBounds
            (new java.awt.Rectangle(115, 54,
                89, 28));
        textFieldsPanel.add (portTextField);
        portTextField.addActionListener (
            new ActionListener() {
                public void actionPerformed
(ActionEvent evt) {
                    conActionPerformed (evt);
                }
            });
        btnPanel.setPreferredSize (new

```

```

java.awt.Dimension(342, 52));
        this.getContentPane ().add (btnPanel,
BorderLayout.SOUTH);
        connectBtn.setText ("Connect");
        btnPanel.add (connectBtn);
        connectBtn.addActionListener (
            new ActionListener() {
                public void actionPerformed
(ActionEvent evt) {
                    conActionPerformed (evt);
                }
            });
    } catch (Exception e) {
        e.printStackTrace ();
    }
}

/**
 * Event handler - invoked when JTextField portTextField,
 * JTextField
 * addressTextField or JButton connectBtn are activated.
 *
 * @param evt Event to be handled.
 */
protected void conActionPerformed (ActionEvent evt) {
    makeConnection ();
    closeDialog ();

    switch (switchKey) {
        case 1 :
            getRemoteFileList ();

            break;

        case 2 :
            sendFile ();

            break;

        case 3 :
            getRemoteFile ();

        default :
            break;
    }
}

```

```

/**
 * Method sendFile sends the specified file to the server.
 */
protected void sendFile () {
    try {
        bufferedOutput.write (
            "sendingFile" + fileName.replaceFirst
                (".csv", ".bsf") + "\n");

        bufferedOutput.flush ();

        encodeFile ();
        receivedMsgs ();

        bufferedOutput.write (base64EncodedText);

        bufferedOutput.flush ();

        closeConnection ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

/**
 * Method getRemoteFileList is used to retrieve a file list
 * from the server.
 */
protected void getRemoteFileList () {
    try {
        bufferedOutput.write ("gettingFileList");
        bufferedOutput.newLine ();
        bufferedOutput.flush ();

        receivedMsgs ();
        closeConnection ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

```

```

    /**
     * Method closeConnection terminates the connection and
     * releases the unused resources.
     *
     * @throws IOException Exception thrown by method
     * closeConnection.
     */
    protected void closeConnection () throws IOException {
        Client.setServerCommTextArea ("Connection
Terminated\n");
        bufferedOutput.close ();
        bufferedInput.close ();
        conn.close ();
    }

    /**
     * Method closeDialog closes the connection dialog box.
     */
    protected void closeDialog () {
        try {
            finalize ();
            dispose ();
        } catch (Throwable e) {
            e.printStackTrace ();
        }
    }

    /**
     * Method makeConnection is used to establish the I/O
     * streams used to communicate with the server.
     */
    protected void makeConnection () {
        try {
            ip = new
String(addressTextField.getText ());
            port = Integer.parseInt
(portTextField.getText ());
            conn = new Socket(ip, port);
            bufferedInput = new BufferedReader(
                new
InputStreamReader(conn.getInputStream ());
            bufferedOutput = new BufferedWriter(
                new
OutputStreamWriter(conn.getOutputStream ());
        } catch (UnknownHostException e) {
            e.printStackTrace ();
        } catch (IOException e) {

```

```

        e.printStackTrace ();
    }

    Client.setServerCommTextArea (
        "Attempting to Connect to IP " + ip + " on port "
+ port +
        ".\nPlease Standby!\n");
    }

    /**
     * Method receivedMsgs is used to process incoming message
     * traffic from the server.
     */
    protected void receivedMsgs () {
        try {
            switch (switchKey) {
                case 1 : //Get file list from server.
                    msg = new String();
                    msgBuffer = new StringBuffer();

                    for (int i = 0; i < 3; i++) {
                        msg = bufferedInput.readLine ();

                        msgBuffer.append (msg);
                    }

                    this.msg = msgBuffer.toString ();
                    Client.setGetFileTextArea
                        (msg.replaceAll (";",
                            "\n"));

                    break;

                case 2 : //Send file to server.
                    msg = bufferedInput.readLine ();
                    Client.setServerCommTextArea ("\n" +
msg);

                    break;

                case 3 : //Get file from server.
                    msg = new String();
                    msgBuffer = new StringBuffer();

                    while ((msg = bufferedInput.readLine
(
)) != null) {
                        msgBuffer.append (msg + "\n");

```

```

    }

    Client.setGetFileTextArea ( "");
    Client.setGetFileTextArea
        (msgBuffer.toString () +
         "\n");

    default :
        break;
    }
} catch (IOException e) {
    e.printStackTrace ();
}

/**
 * Method encodeFile transforms the standard text into
 * Base64 encoded text to send to the server.
 */
protected void encodeFile () {
    base64EncodedText = new
sun.misc.BASE64Encoder().encode (
        fileText.getBytes ());
}

/**
 * Method getRemoteFile sends the request to the server and
 * processes the
 * results from the server.
 */
protected void getRemoteFile () {
    try {
        bufferedOutput.write ("gettingFile" + fileToGet);
        bufferedOutput.newLine ();
        bufferedOutput.flush ();

        receivedMsgs ();

    } catch (IOException e) {
        e.printStackTrace ();
    }
}
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C – SERVER CLASS

```
package server;

import sun.misc.BASE64Decoder;

import java.awt.*;
import java.awt.event.*;

import java.io.*;

import java.net.*;

import javax.swing.*;

/**
 * Class Server instantiates a server that is used to store
 * Base64 encoded files or retrieve and decode Base64 encoded
 * files for viewing by clients.
 *
 * @author David Lowery
 * @version 1.0
 */
public class Server extends javax.swing.JFrame {
    //~ Static fields/initializers -----

    public static JTextArea          statusTextArea;
    private static ServerSocket      encServer;
    private static Socket            con;
    private static BufferedWriter    output;
    private static BufferedReader    input;
    private static String            clientMsg;
    private static String            fileName;
    private static String            fileToStore;
    private static FileOutputStream fileWriter;
    private static StringBuffer      clientMsgBuffer;
    private static int               switchKey;
    private static String            base64EncodedText;
    private static String            cipherText;
    private static String            clearText;
```



```

//~ Instance fields -----

private JScrollPane statusScrollPane;
private JMenuItem  exitMenuItem;
private JMenu       jMenu3;
private JMenuBar    jMenuBar1;

//~ Constructors -----

/**
 * Default Constructor for Server.  Creates a new Server
 * object.
 */
public Server () {
    initGUI ();
}

//~ Methods -----

/**
 * Method initGUI initializes the server GUI.
 */
public void initGUI () {
    try {
        statusScrollPane      = new JScrollPane();
        statusTextArea        = new JTextArea();

        BorderLayout thisLayout = new BorderLayout();
        this.getContentPane ().setLayout (thisLayout);
        thisLayout.setHgap (0);
        thisLayout.setVgap (0);
        this.setDefaultCloseOperation
(WindowConstants.EXIT_ON_CLOSE);
        this.setTitle ("Encryption Server");
        this.setSize (new java.awt.Dimension(807, 627));
        this.setLocation (new java.awt.Point(475, 150));
        this.setIgnoreRepaint (false);
        statusScrollPane.setIgnoreRepaint (false);
        this.getContentPane ().add
            (statusScrollPane,
             BorderLayout.CENTER);
        statusTextArea.setLineWrap (true);
        statusTextArea.setEditable (true);
        statusTextArea.setToolTipText (
            "Communications between server and client are
" + " displayed here.");
        statusScrollPane.add (statusTextArea);
    }
}

```

```

        statusScrollPane.setViewportView
(statusTextArea);
        jMenuBar1          = new JMenuBar();
        jMenu3              = new JMenu();
        exitMenuItem        = new JMenuItem();
        setJMenuBar (jMenuBar1);
        jMenu3.setText ("File");
        jMenu3.setVisible (true);
        jMenuBar1.add (jMenu3);
        exitMenuItem.setText ("Exit");
        exitMenuItem.setVisible (true);
        exitMenuItem.setBounds (new java.awt.Rectangle(5,
5, 60, 30));

        jMenu3.add (exitMenuItem);
        exitMenuItem.addActionListener (
            new ActionListener() {
                public void actionPerformed
(ActionEvent evt) {
                    exitMenuItemActionPerformed (evt);
                }
            });
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }

    /**
     * Method main initializes the server by calling method
     * showGUI.
     *
     * @param args Default parameter for method main.
     */
    public static void main (String [] args) {
        showGUI ();
        runServer ();
    }

    /**
     * Method runServer is the primary method that processes
     * connections with clients.
     */
    private static void runServer () {
        try {
            encServer = new ServerSocket(1969);

            while (true) {
                awaitConnection ();

```

```

        getStreams ();
        whichServiceWasRequested ();
        closeConnection ();
    }
} catch (EOFException eofE) {
    System.out.println ("Connection terminated by
client.");
} catch (IOException e) {
    e.printStackTrace ();
}
}

/**
 * Method closeConnection terminates the connection and
 * releases the unused resources.
 *
 * @throws IOException Exception thrown by method
 * closeConnection.
 */
private static void closeConnection () throws IOException {
    statusTextArea.append ("Connection Terminated\n");
    output.close ();
    input.close ();
    con.close ();
}

/**
 * Method whichServiceWasRequested determines which type of
 * request was made by the client.
 */
private static void whichServiceWasRequested () {
    try {
        clientMsg = new String();

        String msg = "Connection successful... processing
request.\n";

        output.write (msg + "\n");

        output.flush ();

        clientMsg = input.readLine ();

        if (clientMsg.startsWith ("gettingFileList")) {
            sendFileListToClient ();
        } else if (clientMsg.startsWith ("gettingFile"))
{

```

```

        sendFileToClient ();
    } else if (clientMsg.startsWith ("sendingFile"))
    {
        fileName = clientMsg.substring (11);
        processEncodedFile ();
    }
} catch (IOException e) {
    e.printStackTrace ();
}
}

```

```

/**
 * Method sendFileToClient determines which file is to be
 * decoded and returned to the client.
 */

```

```

private static void sendFileToClient () {
    fileName = clientMsg.substring (11);
    decode (fileName);

    try {
        output.write (clearText);

        output.flush ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

```

```

/**
 * Method decode is used to decode the selected Base64
 * encoded file.
 *
 * @param fileName2 File to be decoded.
 *
 * @return String representation of the decoded file.
 */

```

```

public static String decode (String fileName2) {
    try {
        BufferedReader in      = new BufferedReader(
                                new FileReader(fileName2));
        StringBuffer  buffer = new StringBuffer();
        String        text   = new String();

        while ((text = in.readLine ()) != null) {
            buffer.append (text + "\n");
        }
    }
}

```

```

    }

    in.close ();
    base64EncodedText      = buffer.toString ();

    clearText = new String(
        new BASE64Decoder().decodeBuffer
(base64EncodedText));
    } catch (IOException e1) {
        e1.printStackTrace ();
    }

    return clearText;
}

/**
 * Method sendFileListToClient generates a file list from
 * the servers root directory and sends it to the client.
 */
private static void sendFileListToClient () {
    String      files          = ".";
    String []   filesToList    = new File(files).list ();
    StringBuffer fileListBuffer = new StringBuffer();

    for (int i = 0; i < filesToList.length; i++) {
        fileListBuffer.append (filesToList [i] + ";;");
    }

    try {
        output.write (fileListBuffer.toString ());
        output.newLine ();
        output.flush ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

/**
 * Method processEncodedFile captures all of the Strings
 * that represent the file from the client as a single
 * String, which is then passed as an
 * argument to the method storeFile.
 */
private static void processEncodedFile () {
    clientMsgBuffer      = new StringBuffer();

    clientMsg = new String();

```

```

        try {
            while (input.ready ()) {
                clientMsg = input.readLine ();

                clientMsgBuffer.append (clientMsg + "\n");
            }

            fileToStore = clientMsgBuffer.toString ().trim

        };

        statusTextArea.append ("Saving " +
                                fileName + ":\n" +
                                fileToStore);
        storeFile (fileToStore);
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

/**
 * Method storeFile stores the file received from the
 * client.
 *
 * @param fileToStore2 String representation of the Base64
 * encoded file to store on the server.
 */
private static void storeFile (String fileToStore2) {
    try {
        FileWriter fileOut = new FileWriter(fileName);
        fileOut.flush ();
        fileOut.write (fileToStore2);
        fileOut.flush ();
        fileOut.close ();
    } catch (IOException e) {
        e.printStackTrace ();
    }
}

/**
 * Method getStreams gets the I/O streams for communication
 * between client and server.
 *
 * @throws IOException DOCUMENT ME!
 */
private static void getStreams () throws IOException {
    output = new BufferedWriter(
        new OutputStreamWriter(con.getOutputStream

```

```

    ));

    output.flush ();

    input = new BufferedReader(
        new InputStreamReader(con.getInputStream
    ));

    statusTextArea.append ("\nGot Streams\n");

    /**
     * Method awaitConnection tells the server to wait for a
     * connection.
     *
     * @throws IOException Exception thrown by method
     * waitConnection.
     */
    private static void awaitConnection () throws IOException {
        statusTextArea.append ("Waiting for client
connection.\n");
        con = encServer.accept ();
        statusTextArea.append (
            "Connection established from: " +
            con.getInetAddress ().getHostName ());
    }

    /**
     * Method showGUI creates a new instance of this class and
     * shows it inside a new JFrame.
     */
    public static void showGUI () {
        try {
            Server inst = new Server();
            inst.setVisible (true);
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
}

```

```

    /**
     * Event handler - invoked when JMenuItem exitMenuItem from
     * Menu is selected. Terminates client application.
     *
     * @param evt Event to be handled.
     */
    protected void exitMenuItemActionPerformed (ActionEvent
evt) {
        System.exit (0);
    }
}

```


THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D – PARSER WEB SERVICE CLASS

```
package xmlService;

import org.apache.soap.*;
import org.apache.soap.rpc.*;

import java.io.*;

import java.net.*;

import java.util.*;

/**
 * Class Parser provides limited interaction with the specified
 * server via HTTP using SOAP requests. The Parser class
 * retrieves a decoded file from the server and converts it to
 * valid, well formed XML. This class is designed
 * to be ran as a stand-alone client to the server Web service
 * or can be deployed as a Web service itself. This code is a
 * highly modified version of a SOAP test client provided by LT
 * Scott Rosetti/USN.
 *
 * @author David Lowery
 * @version 1.0
 */
public class Parser {
    //~ Instance fields -----

    private String target = "urn:EncServer";
```

```

//~ Constructors -----

/**
 * Default constructor for the Parser.  Creates a new
 * Parser object.
 */
public Parser () {

}

//~ Methods -----

/**
 * Method getDecodedFile makes a SOAP request to the server
 * Web service invoking the "decode" method.  Once
 * retrieved, the text file is parsed and converted into
 * XML.
 *
 * @param str String representation of the URL for the SOAP
 * RPC router.
 * @param file String representation of the filename of the
 * file to be parsed.
 *
 * @return A String representation of the file transformed
 * into XML.
 *
 * @throws Exception Exception thrown.
 */
public String getDecodedFile (String str, String file)
    throws Exception {
    String          value          = null;
    URL             url            = new URL(str);
    int             count          = 0;
    String          filename       = file;
    ArrayList       tokenizedValue = new ArrayList();
    ListIterator    i;
    StringTokenizer tokenizer;

    try {
        Call call = new Call();
        call.setTargetObjectURI (target);
        call.setEncodingStyleURI
(Constants.NS_URI_SOAP_ENC);

        call.setMethodName ("decode");

        Vector params = new Vector();

```

```

        params.addElement (
            new Parameter("filename2", String.class,
filename, null));
        call.setParams (params);

        org.apache.soap.rpc.Response resp = call.invoke
(url, "");

        // Check the response.
        if (resp.generatedFault ()) {
            Fault fault = resp.getFault ();
            System.out.println ("The call failed: ");
            System.out.println ("Fault Code   = " +
                                fault.getFaultCode ());
            System.out.println (
                "Fault String = " + fault.getFaultString
());
        }

        Parameter result = resp.getReturnValue ();

        //System.out.println(result);
        value = (String) result.getValue ();
    } catch (Exception e) {
        System.out.println ("ERROR: " + e);
    }

    tokenizer = new StringTokenizer(value, ", \n\r\t");

    while (tokenizer.hasMoreTokens ()) {
        tokenizedValue.add (tokenizer.nextToken ());
    }

    i = tokenizedValue.listIterator ();

    while (i.hasNext ()) {
        int    index    = i.nextIndex ();
        String element = (String) tokenizedValue.get
(index);

        int    key      = (int) ((i.nextIndex () + 1) %
4);

        //                System.out.println(key);
        switch (key) {
            case 0 :
                tokenizedValue.set (
                    index,

```

```

        "<DutyStatus>" + element +

"</DutyStatus></IndividualStatus>");

        //System.out.println(tokenizedValue.get(index));
        break;

        case 1 :
            tokenizedValue.set (
                index,
                "<IndividualStatus><LastName>" +
element +
                "</LastName>");

        // System.out.println(tokenizedValue.get(index));
        break;

        case 2 :
            tokenizedValue.set (
                index, "<FirstName>" + element +
"</FirstName>");

        // System.out.println(tokenizedValue.get(index));
        break;

        case 3 :
            tokenizedValue.set (
                index, "<Gender>" + element +
"</Gender>");

        // System.out.println(tokenizedValue.get(index));
        break;

        default :
            break;
    }

    i.next ();
}

String st = tokenizedValue.toString ();
st = "<?xml version=\"1.0\"?><PersonnelStatus " +
    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\" " +

    "xsi:noNamespaceSchemaLocation=\"file:///C:/eclipse" +
    "/workspace/Thesis/PersonnelStatusFile.xsd\">" +

```

```

        st.replace ('[' , ' ').replace (',', ' ').replace
(']', ' ') +
        "</PersonnelStatus>";

    return st;
}

/**
 * Method main initializes the Parser and sets the initial
 * parameters for the SOAP client.
 *
 * @param args Default parameter for method main.
 *
 * @throws Exception Exception thrown.
 */
public static void main (String [] args) throws Exception {
    String filename =
        "c:/eclipse/workspace/Thesis/PersonnelStatusFile.
bsf";
    String convertedFile = null;

    try {
        String server = new String(
"http://localhost:8080/soap/servlet/rpcrouter");
        Parser client = new Parser();
        String results = client.getDecodedFile
(server, filename)
        .toString ();
        File file = new
File("PersonnelStatusFile.xml");
        BufferedWriter writer = new BufferedWriter(new
FileWriter(file));

        writer.write (results);
        System.out.println ("File Processed");
        writer.close ();
    } catch (Exception e) {
        System.out.println ("ERROR: " + e);
    }
}
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E – TRANSFORMATION WEB SERVICE CLASS

```
package xmlService;

import org.apache.soap.*;
import org.apache.soap.rpc.*;

import java.io.*;

import java.net.*;

import java.util.*;

/**
 * Class Parser provides limited interaction with the specified
 * server via HTTP using SOAP requests. The Parser class
 * retrieves a decoded file from the
 * server and converts it to valid, well formed XML. This class
 * is designed to be ran as a stand-alone client to the server
 * Web service or can be deployed as a Web service itself. This
 * code is a highly modified version
 * of a SOAP test client provided by LT Scott Rosetti/USN.
 *
 * @author David Lowery
 * @version 1.0
 */
public class Parser {
    //~ Instance fields -----

    private String target = "urn:EncServer";

    //~ Constructors -----

    /**
     * Default constructor for the Parser. Creates a new
     * Parser object.
     */
    public Parser () {
    }
}
```



```

//~ Methods -----

/**
 * Method getDecodedFile makes a SOAP request to the server
 * Web service invoking the "decode" method. Once
 * retrieved, the text file is parsed and converted into
 * XML.
 *
 * @param str String representation of the URL for the SOAP
 * RPC router.
 * @param file String representation of the filename of the
 * file to be parsed.
 *
 * @return A String representation of the file transformed
 * into XML.
 *
 * @throws Exception Exception thrown.
 */
public String getDecodedFile (String str, String file)
    throws Exception {
    String          value          = null;
    URL             url            = new URL(str);
    int             count          = 0;
    String          filename       = file;
    ArrayList       tokenizedValue = new ArrayList();
    ListIterator    i;
    StringTokenizer tokenizer;

    try {
        Call call = new Call();
        call.setTargetObjectURI (target);
        call.setEncodingStyleURI
(Constants.NS_URI_SOAP_ENC);

        call.setMethodName ("decode");

        Vector params = new Vector();
        params.addElement (
            new Parameter("filename2", String.class,
filename, null));
        call.setParams (params);

        org.apache.soap.rpc.Response resp = call.invoke
(url, "");
    }
}

```

```

        // Check the response.
        if (resp.generatedFault ()) {
            Fault fault = resp.getFault ();
            System.out.println ("The call failed: ");
            System.out.println ("Fault Code    = " +
fault.getFaultCode ());
            System.out.println (
                "Fault String = " + fault.getFaultString
            ());
        }

        Parameter result = resp.getReturnValue ();

        //System.out.println(result);
        value = (String) result.getValue ();

    } catch (Exception e) {
        System.out.println ("ERROR: " + e);
    }

    tokenizer = new StringTokenizer(value, ", \n\r\t");

    while (tokenizer.hasMoreTokens ()) {
        tokenizedValue.add (tokenizer.nextToken ());
    }

    i = tokenizedValue.listIterator ();

    while (i.hasNext ()) {
        int    index    = i.nextIndex ();
        String element = (String) tokenizedValue.get
(index);

        int    key      = (int) ((i.nextIndex () + 1) %
4);

        //          System.out.println(key);
        switch (key) {
            case 0 :
                tokenizedValue.set (
                    index,
                    "<DutyStatus>" + element +

"</DutyStatus></IndividualStatus>");

                //System.out.println(tokenizedValue.get(index));
                break;

```

```

        case 1 :
            tokenizedValue.set (
                index,
                "<IndividualStatus><LastName>" +
element +
                "</LastName>");

            //System.out.println(tokenizedValue.get(index));
            break;

        case 2 :
            tokenizedValue.set (
                index, "<FirstName>" + element +
"</FirstName>");

            //System.out.println(tokenizedValue.get(index));
            break;

        case 3 :
            tokenizedValue.set (
                index, "<Gender>" + element +
"</Gender>");

            //System.out.println(tokenizedValue.get(index));
            break;

        default :
            break;
    }

    i.next ();
}

String st = tokenizedValue.toString ();
st = "<?xml version=\"1.0\"?><PersonnelStatus " +
    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\" " +
    "xsi:noNamespaceSchemaLocation=\"file:///C:/eclipse" +
    "/workspace/Thesis/PersonnelStatusFile.xsd\">" +
    st.replace ('[', ' ').replace (',', ' ').replace
(']', ' ') +
    "</PersonnelStatus>";

    return st;
}

```

```

    /**
     * Method main initializes the Parser and sets the initial
     * parameters for the SOAP client.
     *
     * @param args Default parameter for method main.
     *
     * @throws Exception Exception thrown.
     */
    public static void main (String [] args) throws Exception {
        String filename      =
"c:/eclipse/workspace/Thesis/PersonnelStatusFile.bsf";
        String convertedFile = null;

        try {
            String          server = new String(
"http://localhost:8080/soap/servlet/rpcrouter");
            Parser          client  = new Parser();
            String          results = client.getDecodedFile
(server, filename)
                                .toString ();
            File            file    = new
File("PersonnelStatusFile.xml");
            BufferedWriter writer = new BufferedWriter(new
FileWriter(file));

            writer.write (results);
            System.out.println ("File Processed");
            writer.close ();

        } catch (Exception e) {
            System.out.println ("ERROR: " + e);
        }
    }
}

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F – PHP CLIENTS

The PHP code used to call the Web services from the Web site was very simple to implement and only consisted of a few lines. The following constitutes the three different clients implemented to call the services provided by the POC.

The “Raw Data” client:

```
<?php

require 'SOAP/Client.php';

$soap = new
    SOAP_Client('http://localhost:8080/soap/servlet/rpcrouter');

$params = new SOAP_Value('file', 'string',
    'c:/eclipse/workspace/Thesis/PersonnelStatusFile.bsf');

$hits = $soap->call('decode', $params, 'urn:EncServer');

print_r($hits);

?>
```

The “XML Data” client:

```
<?php

require 'SOAP/Client.php';

$soap = new
    SOAP_Client('http://localhost:8080/soap/servlet/rpcrouter');

$params = array(
    new SOAP_Value('url', 'string',
        'http://localhost:8080/soap/servlet/rpcrouter'),
    new SOAP_Value('file', 'string',
        'c:/eclipse/workspace/Thesis/PersonnelStatusFile.bsf')
);

$xmlFile = $soap->call('getDecodedFile', $params,
    'urn:XMLPersonnelStatusService');
```

```
print_r($xmlFile);  
?>
```

The “Summary Report” client:

```
<?php  
  
require 'SOAP/Client.php';  
  
$soap = new  
    SOAP_Client('http://localhost:8080/soap/servlet/rpcrouter');  
  
$params = array(  
    new SOAP_Value('url', 'string',  
        'http://localhost:8080/soap/servlet/rpcrouter'),  
    new SOAP_Value('file', 'string',  
        'c:/eclipse/workspace/Thesis/PersonnelStatusFile.bsf')  
);  
  
$htmlFile = $soap->call('getTransformedFile', $params,  
    'urn:PersonnelStatusSummary');  
  
print_r($htmlFile);  
?>
```

BIBLIOGRAPHY

- Achour, Mehdi. Friedhelm Betz, Antony Dovgal, Nuno Lopes, Philip Olson, Georg Richter, Damien Seguy and Jakub Vrana, “PHP Manual”, PHP.net, 28 August 2004, <http://www.php.net/manual/en/>, Last Accessed: September 2004.
- “Apache SOAP v2.3.1 Documentation”, Apache SOAP, <http://ws.apache.org/soap/docs/index.html>, Last Accessed: September 2004.
- Ballinger, Keith. David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham, Prasad Yendluri, “Basic Profile 1.1”, WS-I, 24 August 2004, <http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>, Last Accessed: September 2004.
- Boubez, Toufic. Glen Daniels, Doug Davis, Steve Graham, Yuichi Nakamura, Ryo Neyama and Simeon Simeonov. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Sams Publishing, 2001.
- Brittain, Jason and Ian F. Darwin. Tomcat: The Definitive Guide. O'Reilly, 2003.
- Bryan, Douglas. Vadim Draluk, Dave Ehnebuske, Tom Glover, Andrew Hatley, Yin Leng Husband, Alan Karp, Keisuke Kibakura, Chris Kurt, Jeff Lancelle, Sam Lee, Sean MacRoibeaird, Anne Thomas Manes, Barbara McKee, Joel Munter, Tammy Nordan, Chuck Reeves, Dan Rogers, Christine Tomlinson, Cafer Tosun, Claus von Riegen and Prasad Yendluri. “UDDI Version 2.04 API Specification”, UDDI Committee, 19 July 2002, <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>, Last Accessed: September 2004.
- Bueno, Ricardo, “Global Information Grid – Questions and Answers”, The DoD Software Tech News, April 2004, 8 – 9.
- Burke, Eric M., Java and XSLT. O'Reilly, 2001.
- Chappell, Dave and Tyler Jewell. Java Web Services. O'Reilly, 2002.
- “Core Enterprise Services Strategy – Draft Version 1.1”, Office of the Assistant Secretary of Defense for Networks and Information Integration, 1 July 2003.
- Cornell, Gary and Cay S. Horstmann. Core Java 2: Volume II – Advanced Features. Prentice Hall PTR, 2001.
- Coyle, Frank P., XML, Web Service, and the Data Revolution. Addison Wesley, 2002.

- Drager, Steve. Lois Walsh, Dr.. “Grid Computing for Information Superiority”, Software Tech News, Volume 7, Number 1, <http://www.softwaretechnews.com/stn7-1/grid-superiority.html>, Last Accessed: September 2004.
- Farrell, Stephen and Shivaram Mysore (Working Group Chairs). “Mission Statement”, XML Key Management Working Group, <http://w3.org/2001/XKMS/#Mission>, Last Accessed: September 2004.
- Foster, Ian and Lee Liming, “The Role of Standards in the Grid”, The DoD Software Tech News, April 2004, 10-14.
- “Grid Computing Defined”, Software Tech News, Volume 7, Number 1, <http://www.softwaretechnews.com/stn7-1/grid-defined.html>, Last Accessed: September 2004.
- Harold, Elliotte Rusty. Java I/O. O’Reilly, 1999.
- Harold, Elliotte Rusty. Java Network Programming, 2nd Edition. O’Reilly, 2000.
- Haas, Hugo. Web Services Glossary, W3C – World Wide Web Consortium, 11 February 2004, <http://w3.org/tr/2004/note-ws-gloss-20040211/>, Last Accessed: September 2004.
- Hutton, Claude O. Jr., “3D Battelspace Visualization Using Operational Planning Data”, Master’s Thesis, Naval Postgraduate School, 2003.
- J2EE Documentation, Sun Java, <http://java.sun.com/j2ee/1.4/docs/index.html>, Last Accessed: September 2004.
- J2SE Documentation, Sun Java, <http://java.sun.com/j2se/1.4.2/docs/index.html>, Last Accessed: September 2004.
- JWSDP, Sun Java, <http://java.sun.com/webservices/jwsdp/index.jsp>, Last Accessed: September 2004.
- Knudsen, Jonathan and Patrick Niemeyer. Learning Java, 2nd Edition. O’Reilly, 2002.
- Mangano, Sal. XSLT Cookbook. O’Reilly, 2002.
- Merriam-Webster Online, <http://m-w.com>, Last Accessed: September 2004.
- Metsker, Steven John. Building Parsers with Java. Addison Wesley, 2001.
- Meyerriecks, Dawn. “Net-Centric Enterprise Services: What Problem Are We Trying To Solve?”, Global Information Grid Enterprise Services, 9 August 2001, <http://ges.dod.mil/articles/netcentric.htm>, Last Accessed: September 2004.

- Microsoft, “COM: Delivering on the Promises of Component Technology”, MicrosoftCOM, <http://www.microsoft.com/com/>, Last Accessed: September 2004.
- Net-Centric Warfare, <http://www.dtic.mil/jcs/j6/education/warfare.html>, Last Accessed: September 2004.
- OASIS – Organization for Advancement of Structured Information Standards, <http://oasis-open.org>, Last Accessed: September 2004.
- Object Management Group, “CORBA Basics”, CORBA FAQ, <http://www.omg.org/gettingstarted/corbafaq.htm>, Last Accessed: September 2004.
- Osias, Michael, “Grid Computing – A Service Perspective”, The DoD Software Tech News, April 2004, 5 – 8.
- Reagle, Joseph (Working Group Chair). “XML Encryption Charter”, XML Encryption Working Group, June 2002, <http://w3.org/Encryption/2002/06/xmlenc-charter>, Last Accessed: September 2004.
- Reilly, David and Michael Reilly. Java Network Programming and Distributed Computing. Addison Wesley, 2002.
- Sklar, David and Adam Trachtenberg. PHP Cookbook. O’Reilly, 2002.
- Sridharan, Prashant and Richard Steflik. Advanced Java Networking. Prentice Hall PTR, 2000.
- “The Challenge”, Global Information Grid Enterprise Services, <http://ges.dod.mil/about/challenge.htm>, Last Accessed: September 2004.
- The Open Group, “What is Distributed Computing and DCE?”, DCE Portal, <http://www.opengroup.org/dce>, Last Accessed: September 2004.
- “The Resources”, Global Information Grid Enterprise Services, <http://ges.dod.mil/about/resources.htm>, Last Accessed: September 2004.
- “The Solution”, Global Information Grid Enterprise Services, <http://ges.dod.mil/about/solution.htm>, Last Accessed: September 2004.
- Thompson, Laura and Luke Welling. PHP and MySQL Web Development, Second Edition. Sams Publishing, 2003.
- UDDI – Universal Description, Discovery and Integration, <http://www.uddi.org/>, Last Accessed: September 2004.

Van Dine, Wayne A. Jr., “GIG from a Warfighter Perspective”, 27 October 2003, Email attachment from Ricardo Bueno – a member of the GIG ES Architecture Working Group, among others.

Varhol, Peter. “SOA Offers Competitive Advantages”, FTPOnline, 30 March 2004, http://www.ftponline.com/special/soa/overview/default_pf.asp, Last Accessed: September, 2004.

Vaswani, Vikram. XML and PHP. New Riders Publishing, 2002.

W3C – World Wide Web Consortium, <http://w3.org/>, Last Accessed: September 2004.

Walker, Ellen, “Grid Computing” , The DoD Software Tech News, April 2004, 3-14.

Web Services Description Working Group, “Web Services Description Language Version 2.0 Part 1: Core Language”, 3 August 2004, <http://www.w3.org/TR/2004/WD-wsdl20-20040803>, Last Accessed: September

Webopedia, <http://www.webopedia.com>, Last Accessed: September 2004.

WS-I – Web Services Interoperability, <http://www.ws-i.org/>, Last Accessed: September 2004.

XML Activity, “Extensible Markup Language”, World Wide Web Consortium, <http://w3.org/XML/>, Last Accessed: September 2004.

XML Activity, “Extensible Markup Language”, World Wide Web Consortium, <http://w3.org/XML/>, Last Accessed: September 2004.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, VA
2. Dudley Knox Library
Naval Postgraduate School
Monterey, CA
3. Marine Corps Representative
Naval Postgraduate School
Monterey, CA
4. Directors, Training and Education
MCCDC, Code C46
Quantico, VA
5. Director, Marine Corps Research Center
MCCDC, Code C40RC
Quantico, VA
6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
Camp Pendleton, CA
7. Dr. Sue Numrich
Defense Modeling and Simulation Office (DMSO)
Alexandria, VA
8. Virginia Dobey
Defense Modeling and Simulation Office (DMSO)
Alexandria, VA
9. Mike Rugienius
Defense Modeling and Simulation Office
Alexandria, VA
10. Chris Turrell
Defense Modeling and Simulation Office (DMSO)
Alexandria, VA
11. COL George Stone
Battle Command, Simulation & Experimentation Directorate (DAMO-SB)
Crystal City, VA

12. MAJ Favio Lopez
Battle Command, Simulation & Experimentation Directorate (DAMO-SB)
Crystal City, VA
13. Robert Eubank
NSA
Ft. Meade, MD
14. Associate Professor Don Brutzman
Naval Postgraduate School
Monterey, CA
15. Research Associate Curt Blais
Naval Postgraduate School
Monterey, CA
16. Research Associate Jeff Weekley
Naval Postgraduate School
Monterey, CA
17. Capt James Neushul
MCTSSA
Camp Pendleton, CA